A survey on energy estimation and power modeling schemes for smartphone applications

Raja Wasim Ahmad | Abdullah Gani | Siti Hafizah Ab Hamid | Mohammad Shojafar | Abdemutlib Ibrahim Abdalla Ahmed | Sajjad A. Madani | Kashif Saleem | Joel J.P.C. Rodrigues

1 Center for Mobile Cloud Computing Research (C4MCCR), FSKTM, University of Malaya, Kuala Lumpur, Malaysia
2 The Sapienza University of Rome, Rome, Italy
3 Department of Computer Science, COMSATS Institute of Information Technology, Islamabad, Pakistan
4 Center of Excellence in Information Assurance (CoEIA), King Saud University (KSU), Riyadh, Saudi Arabia
5 National Institute of Telecommunications (INatel), Santa Rita do Sapucaí, MG, Brazil
6 University ITMO, St. Petersburg, Russia

Correspondence
Raja Wasim Ahmad and Abdullah Gani, Center for Mobile Cloud Computing Research (C4MCCR), FSKTM, University of Malaya, Kuala Lumpur, Malaysia.
Email: abdullah@um.edu.my; wasimraja@siswa.um.edu.my

Funding information
Finep, 01.14.0231.00. Government of Russian Federation, 074-U01. UM.C/625/1/HIR/MOE/ FCSIT/03Malaysian Ministry of Education.

Summary
In the last decade, the rising trend in the popularity of smartphones motivated software developers to increase application functionality. However, increasing application functionality demands extra power budget that as a result, decreases smartphone battery lifetime. Optimizing energy critical sections of an application creates an opportunity to increase battery lifetime. Smartphone application energy estimation helps investigate energy consumption behavior of an application at diversified granularity (e.g., coarse and fine granular) for optimal battery resource use. This study explores energy estimation and modeling schemes to highlight their advantages and shortcomings. It classifies existing smartphone application energy estimation and modeling schemes into 2 categories, i.e., code analysis and mobile components power model–based estimation owing to their architectural designs. Moreover, it further classifies code analysis–based modeling and estimation schemes in simulation-based and profiling-based categories. It compares existing energy estimation and modeling schemes based on a set of parameters common in most literature to highlight the commonalities and differences among reported literature. Existing application energy estimation schemes are low-accurate, resource expensive, or non-scalable, as they consider marginally accurate smart battery’s voltage/current sensors, low-rate power capturing tools, and labor-driven lab-setting environment to propose power models for smartphone application energy estimation. Besides, the energy estimation overhead of the components power model–based estimation schemes is very high as they physically run the application on a smartphone for energy profiling. To optimize smartphone application energy estimation, we have highlighted several research issues to help researchers of this domain to understand the problem clearly.

KEYWORDS
application energy, energy estimation, energy profiling, profiling overhead

1 | INTRODUCTION

Nowadays, with the proliferation of portable devices, the energy-efficient system design has become a must-to-meet requirement for recent resource-constrained smartphone devices. The ever-growing smartphone user’s demands encourage application developers to enrich the legacy applications that as a result, increase smartphone’s computational and communication cost. Among all smartphone-based applications, on-demand video,1 multi-agent–based distributed games,2 spatial locality-based social applications,3 pedestrian tracking,4,5 and context-based advertisement services6 are the most energy-hungry services. The inherent features of these services considerably increase the energy consumption of smartphone applications.
demands of processors when executed on smartphones.\textsuperscript{7–9} It is estimated that in last 2 decades, processor power budget has increased significantly because of smartphone application's high resource use. To efficiently exploit smartphone's power budget, software optimization minimizes inter-component interaction among hardware modules that as a result, increases battery's lifetime. However, software optimization methods require highly accurate and resource-friendly energy estimation models to identify the battery-hungry portion within a smartphone application.\textsuperscript{7,8,10–13} smartphone applications,\textsuperscript{15–17} (b) identify system energy consumption,\textsuperscript{18} (c) find per-application energy consumption,\textsuperscript{19} and (d) re-factor application's source code to increase its quality. Smartphone components such as central processing unit (CPU), Wi-Fi, memory, bluetooth, and network radios are the most energy-consuming components. The power consumed by each of the components depends on (a) component's power consumption at idle state, (b) use level, (c) number of modeled power states, (d) transition count among system power states, and (e) power consumption rate for each transition. For each component, idle power state represents minimum power required to keep the component in an active state with minimal workload. Therefore, for a smartphone application, energy consumption is defined as the accumulative sum of power consumed by all the modules within a smartphone during activity time frame. For instance, during network communication, activities of an application such as send data, receive data, and close network socket trigger its components to perform specific task that as a result, deplete battery charge.\textsuperscript{11,15,16,20–29} Smartphone application energy estimation weighs application energy consumption at unlike granularity levels including source code line, routine, thread, process, and execution path. Fine granular source code instruction energy profile creates an opportunity for application developers to investigate application energy consumption at earlier development stages.\textsuperscript{30–33} Within an application's source code, to each instruction, a fixed energy cost (called base cost) is associated depending on the complexity of the operation it performs when executed on a smartphone device. Code analysis–based energy estimation exploits base cost energy of instructions to estimate energy consumption of smartphone applications. Base cost energy of an instruction is estimated either using cycle-accurate simulators or external measurement setup as presented in Figure 2.\textsuperscript{34–36} However, accuracy of energy estimation using code analysis–based methodology highly depends on (a) execution path estimation, (b) instructions storage location prediction, (c) circuit state overhead, and (d) loops bound estimation.\textsuperscript{37–43} To the best of our knowledge, there are only a few studies that surveyed smartphone application energy estimation schemes. One of the studies proposed by Ahmad et al\textsuperscript{44} has considered profiling energy consumption of mobile applications. However, the proposed work lacks in considering application energy estimation in terms of code analysis and smartphone components power model–based energy estimation. Hoque et al\textsuperscript{36} surveyed energy bugs, types/sources of application bugs, debugging techniques, application performance, and online/offline profilers. However, in the reported work, limited analysis on the surveyed estimation schemes is presented. Both of these works overlooked a comprehensive analysis and classification of existing energy estimation schemes. The main contributions of this work include the following:

i. Classifying existing state-of-the-art smartphone application energy estimation schemes based on the thematic taxonomies. Our proposed thematic taxonomies classify state-of-the-art code analysis and mobile components power model–based estimation schemes using a set of parameters common in most literature.

ii. Analyzing current energy estimation schemes (qualitatively and quantitatively) by highlighting their critical aspects and implications.

iii. Proposing and discussing open research issues and challenges to assist researchers in identifying appropriate topics for future research.

The rest of the paper is organized as follows. Section 2 thoroughly describes some necessary background on this domain of research. Section 3 debates on the code analysis–based smartphone application energy schemes. Section 4 discusses mobile component power model–based energy estimation. Section 5 discusses mobile energy efficiency and technological advancements in Information Technology (IT) sector. Section 6 discusses open research issues in this domain of research. Section 7 briefly discusses outcomes of this study. Based on the observations, Section 9 concludes the paper and suggests further research directions. Throughout this paper, we have interchangeably used terminology, ie, mobile, mobile phone, and smartphone, to represent resource-constrained smartphone devices.

## 2 | MOTIVATIONS

Nowadays, smartphone devices are replacing desktop servers as preferences of users to perform computing have changed. In last few years, the trend in the rise of the number of smartphone users has rapidly increased as shown in Figure 1. It is estimated that by 2016, number of smartphone users will increase to 1.82 × 10\textsuperscript{9} in the world because of their increased accessibility, improved usability, and attractive applications features. According to the info-graphic report,
because of increasing rate in application download percentage, by 2017, the smartphone application market will generate $77 billion worth of revenue*. However, despite this tremendous hype in smartphone popularity, still, their use is limited by the design of the battery. Smartphone consumes a significant amount of world electricity budget because of frequent battery charging. According to Barry Fischer report, the amount of energy to charge iPhone 5 smartphones in the world is equivalent to the total energy use of 54,000 US households for 1 year. According to this report, iPhone 5 electricity demand per year is 3.5 to 4.9 kWh. Also, the monetary cost to charge iPhone 5 per year is $0.41. Energy estimation is one of the ways to effectively use a smartphone battery to augment device battery lifetime to minimize total electricity budget of the world.

Smartphone application energy estimation provides the basis for green smartphone computing. During application development process, developers usually consider maintainability, complexity, usability, and understandability as the performance measurement parameters for their applications. It is estimated that 80% of the application developers are not aware of green software development strategies. Energy estimation of smartphone applications provides feedback to the application developers to reconsider their application design for effective battery resource use. Smartphone application energy estimation facilitates to (a) identify resource critical rogue applications, (b) diagnose smartphone energy consumption, (c) estimate per-application energy use, and (d) optimize application design.44,47-51

3 | BACKGROUND

This section discusses the notion of smartphones, smartphone applications, and energy estimation methods.

3.1 | Notion of smartphone application

A smartphone is a cellular device that performs many of the functions of a desktop computer and offers internet access (Wi-Fi and 3G), touch screen interface, local and remote data storage, and operating system (OS) capable of running downloaded applications. It integrates capabilities of a cell phone to more common features of a handheld computer or personal digital assistant to enrich the services for the user. Because of size limitations, smartphone carries low speed CPU limited capacity of random-access memory (RAM) storage. Moreover, smartphones are equipped with lithium battery with less than 300 mAh capacity that lasts for a few hours when the network and compute intensive applications run on it.1,44,52,53

The hype in popularity of smartphones has remarkably increased because of their unlimited applications in various computing domains such as education,54,55 management information system,56 and health monitoring.57,58 Smartphone applications have inherited main features of rich internet applications to enrich user experience. Recent smartphone applications offer code portability, asynchronous communication, quick responsiveness, and extensive functionality. It embodies context-aware and multi-tier services to entertain smartphone users.1,59,60 However, in line with increasing functionality, these applications heavily use light sensors, Global Positioning System, compass, and accelerometers, to perform the desired task. As a result, sensors deplete battery charge when application triggers them to perform required task.1,61,62

3.2 | Smartphone application energy estimation

Smartphone application energy estimation considers power models for smartphone components or software operations to estimate energy consumption of smartphone applications. Smartphone components power model–based energy estimation uses state of charge (SOC) estimation methods (coulomb counting or voltage) to forecast energy consumption of an application as highlighted in Figure 2. Alternatively, software operation’s cost-based estimation method (also known as code analysis–based estimation) considers base cost energy of source code instructions to estimate energy consumption of the smartphone application. Sections 3.2.1 and 3.2.2 briefly discuss energy estimation methodologies for smartphone applications.

3.2.1 | State of charge energy estimation

State of charge represents total remaining charge capacity inside a smartphone battery. It reflects the performance of a battery. Accurate SOC estimation improves battery life, prevents over-discharge, and assists application developers to adopt rational control strategies to save energy.53,64 State of charge for smartphone battery is estimated based on coulomb counting18,65 and terminal voltage method.66-68 Coulomb counting estimates SOC based on an accumulative current drop by directly accessing the current sensor within smartphone devices. The accuracy of coulomb counting method is highly affected by numerous external and internal factors such as battery aging, temperature, and charging/discharging rate.36,69-71 Alternatively, terminal voltage

---

*www.smartinsights.com/mobile-marketing
method estimates SOC based on voltage drop because of the internal impedance of battery during its discharging. Both coulomb counting and terminal voltage methods are implemented in battery fuel gauge hardware module of the smartphone battery. For energy estimation of an application, OS considers Android power application program interfaces (APIs) to access the fuel gauge.\textsuperscript{72,73}

### 3.2.2 Base cost energy estimation

Smartphone application performs a series of activities during its execution on smartphone devices. Base cost energy of an activity truly depends on the circuitry it triggers during execution on the smartphone device. Within an application, each activity such as variable declaration has a fixed energy cost associated with it depending on the degree of interaction with smartphone components at execution time. Energy cost of fine granular instructions is much lower than coarse granular operations (variable declaration, assignment). The energy cost of an instruction within Acorn Reduced Instruction Set Computer Machine (ARM) architecture, called ARM Instruction Set Architecture (ISA), is dissimilar and highly depends on cycles per instruction for a particular instruction. Also, for a single instruction, energy consumption also depends on the type of op-code, a number of operands, operands type, and type of smartphone model. For instance, base cost energy for LOAD instruction is higher than ADD instruction as memory-based operations are expensive than simple arithmetic logic unit–oriented operations. Base cost energy of a program is estimated based on the energy cost of source code that the application is constitute of.\textsuperscript{10,37,74–76}

### 3.3 Smartphone components power consumption

The energy of a smartphone application is estimated based on the power consumption of components of the smartphone. Power consumption rate of each smartphone component is different and highly depends on its execution state during application execution on smartphone. For instance, the power consumption of a CPU component highly depends on its frequency during application execution (dynamic voltage frequency scaling [DVFS] method). Similarly, the power consumption of Wi-Fi module depends on the amount of data transferred and energy per byte transfer. The energy of transferring of each byte of data within application depends on Wi-Fi state during data transfer. At low power state, smartphone components consume less energy than a high power state. For an average smartphone application, CPU consumes most energy budget of application closely followed by Wi-Fi module. For iMate Kjam smartphone, Anand et al\textsuperscript{77} reported that during normal functioning, CPU consumes 35% of the total battery budget. Alternatively, Global System for Mobile, Wi-Fi, backlight, and bluetooth consumed 25%, 25%, 3%, and 7% energy consumption of the total energy budget.

### 4 CODE ANALYSIS–BASED ENERGY ESTIMATION AND MODELING

Code analysis–based estimation analyzes source code of the application and uses instructions energy cost models to estimate energy consumption. For a smartphone application, each instruction exhibits unlike resource consumption; therefore, each instruction has dissimilar CPU execution time and power consumption cost. This section presents thematic taxonomy and debates on current state-of-the-art energy estimation schemes.

### 4.1 Taxonomy of code analysis–based energy estimation and modeling

Figure 3 presents our thematic taxonomy that classifies existing code analysis–based energy estimation schemes into several categories, ie, abstraction level, instruction profiling method, program structure analysis, estimation overhead, profiled instruction type, and estimation granularity.

- Attributes of abstraction level parameter describe the hardware granularity at which energy is profiled to construct fine granular instruction power models.
Abstraction level attributes include architectural and nonarchitectural details. In the case of architectural details attribute, hardware performance counters (HPCs) are used to construct instruction power models. However, for the non-architectural attribute, instruction power consumption is estimated based on the external power measurement tools.

- Instruction profiling method parameter defines the technique opted to construct power models for ARM ISA instructions. For instruction power modeling, instruction profiling method parameter followed source code instrumentation or test program(s) attributes. For source code instrumentation case, test application is instrumented to record the time-stamped power during application execution on a smartphone (online). In offline phase, online generated profile is processed to construct instruction power models. Alternatively, test program–based instruction power modeling runs test programs on a smartphone to capture voltage drop at sense resistor placed at power rail of mobile battery. Test program consists of millions of instructions belonging to the same instruction class.

- Attributes of program structure analysis parameter define the basic ingredients of an application that an estimation scheme considered during energy estimation. Program's base cost represents energy cost of a program based on the individual base cost energy of each instruction within the smartphone application. However, base cost energy does not reflect true estimation behavior because of resource limitation of target smartphone architecture; for instance, relocating neighbor instructions position gives dissimilar energy consumption values. The energy consumption of application truly depends on the path that an application took at dynamic execution time. To find the execution path, smartphone applications are instrumented prior to running on the mobile phone to record execution paths (offline mode).

- Estimation overhead reflects the energy consumption rate of estimation tool to assess energy use rate of a smartphone application. Based on the architectural complexity of estimation tools, estimation overhead is attributed as low or high. Low-value attribute defines that proposed scheme follows lightweight estimation design. Whereas, high attribute defines that proposed scheme does not effectively use underlying mobile phone resources to estimate application energy consumption.

- Source code of the application is of diverse types, ie, C, Java, or assembly, depending on the design of an application. Based on the nature of source code, type of an instruction (whose energy consumption is to be modeled) is different. Profiled instruction type parameter defines the type of instruction that is profiled to estimate application energy consumption. The attributes of profiled instruction type include assembly instructions, system calls, and APIs, as shown in Figure 3. An assembly instruction presents the lowest system details when interacting with a smartphone. On the other hand, some of the code-based analysis scheme considered power profiling at system calls or API level.

- Estimation granularity describes the level to which estimation tool estimates energy consumption. Estimation granularity includes application, line, path, and routines within an application. Application level energy
estimation is most trivial as it requires considering every chunk of application’s source code. The line represents the high-level source code line that is considered during the profiling process (eg, compound assignment statement and increment statement). Execution path is also a complex entity as the execution of a particular path depends on the values of the condition set in comparison statement.

- Look-up table–based energy estimation does not simulate architectural details for mobile application energy estimation. Rather, it uses already profiled architectural details to estimate application energy consumption. However, size of the look-up table greatly impacts estimation time (Table 1).

4.2 | State-of-the-art code analysis–based energy estimation and modeling schemes

This section briefly discusses existing energy estimation schemes that considered code analysis–based estimation methodology to assess energy needs of an application. Based on the design of energy estimation schemes, state-of-the-art code analysis–based estimation schemes are classified into 2 categories including profiling-based energy estimation and simulator-based energy estimation as discussed in the following sections.

4.2.1 | Profiling-based schemes

Profiling-based energy estimation paradigm exploits the external hardware–based laboratory setting environment to construct power models for instructions. In this section, we debate on profiling-based energy estimation schemes to highlight their advantages and shortcomings.

In the work of Tiwari et al,37 to estimate power consumption of embedded software, an instruction power cost–based estimation technique is proposed. The instruction power cost (assembly-based) called base cost energy is estimated by creating and running test programs comprising several instances of the same instruction running inside a tight loop to capture current drop across the battery terminals. However, inside the application instruction, execution order impacts estimation accuracy because of circuit state overhead. To handle the circuit state overhead, inter-instruction effects of a few instructions are modeled. The proposed framework identifies basic building blocks within application followed by per block energy estimation to forecast energy demands of the application. The advantage of the proposed technique is simplicity and applicability towards identifying application energy consumption at earlier development stages. However, the proposed technique requires per CPU cycle energy consumption to model power cost of each assembly-based instruction. However, finding low architectural details is computationally expensive, and also, it requires high sample rate–based power meters. Also, in comparison to sequential-based embedded applications considered in this study, the nondeterministic behavior of mobile application poses numerous challenges that require rigorous program analysis to estimate energy consumption.

To investigate the power consumed by various activities involved in ARM instruction execution, authors proposed a model to break down an instruction’s power consumption into its sub-activities. The proposed model78 calculates ARM ISA power consumption based on the aggregate sum of energy consumed by processor cores, memory controller, and static RAM. The modeled parameters include register bank flip-flops, the number of shift operations, instruction weight, and the hammering distance between instructions. Moreover, the proposed scheme modeled inter-instruction effect for a set of instructions because of resource-constrained nature of mobile phones. To capture the power consumption at mobile component granularity, the proposed approach (similarly to the work of Tiwari et al37) places a low-resistance resistor (1 ohm) at the power supply line of microcontroller. The proposed processor energy model for a single instruction is presented in Equations 1, 2, and 3. In the mentioned equations, \( E_{\text{Control(code)}} \) and \( E_{\text{flash}} \) represent the energy consumed in the memory controller and caused by the code in flash memory, respectively. Alternatively, \( E_{\text{IF}} \), \( E_{\text{ID}} \), and \( E_{\text{EX}} \) represent the energy consumed during fetch, decode, and execute stages of CPU pipelines. Also, \( E_{\text{stall}} \) represents the energy consumption during stalling in pipeline stages.

\[
E_{\text{fetch}} = E_{\text{Control(code)}} + E_{\text{flash}} + E_{\text{IF}} \quad (1)
\]
\[
E_{\text{decode}} = E_{\text{ID}} \quad (2)
\]
\[
E_{\text{execute}} = E_{\text{EX}} + E_{\text{Control(data)}} + E_{\text{flash(data)}} + E_{\text{stall}} \quad (3)
\]

The proposed approach is useful when lower architectural details are required to optimize the design of the application. However, to find energy for each mobile component, the rest of the mobile components is switched off to minimize the effect of estimation noise that as a result, requires extensive offline regression analysis. Also, the proposed approach places 1 ohm resistor at each component power rail to capture current drop that makes proposed methodology suitable and adaptable only for lab experiments.

In the study of Khoshbakht et al,79 to analyze ARM ISA power consumption behavior during varying types of data within store memory operations, a study is conducted. The proposed scheme enhanced Intel Pin cycle-accurate estimation tool to instrument the store instructions. It was concluded that with more bit transitions (number of ‘1’s), the energy consumption of store operation surges significantly. However, this scheme lacks in considering the effect of changing the storage location of data, ie, cache and memory. It also does not relate power consumption behavior of store
<table>
<thead>
<tr>
<th>Reference</th>
<th>Abstraction level</th>
<th>Instruction profiling method</th>
<th>Program structure analysis</th>
<th>Estimation overhead</th>
<th>Estimated instruction type</th>
<th>Estimation granularity</th>
<th>Target processor</th>
<th>Claimed accuracy, %</th>
<th>Benchmark(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>Architectural details</td>
<td>Test programs</td>
<td>Program base cost energy and inter-instructions effects (circuit state overhead)</td>
<td>N/A</td>
<td>Assembly-based instructions</td>
<td>Application</td>
<td>Intel 486DX2</td>
<td>N/A</td>
<td>Synthetic</td>
</tr>
<tr>
<td>79</td>
<td>Non-architectural details</td>
<td>Source code instrumentation</td>
<td>N/A</td>
<td>N/A</td>
<td>Assembly-based instructions</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>Spec2000 CPU</td>
</tr>
<tr>
<td>38</td>
<td>Non-architectural details</td>
<td>Test programs</td>
<td>Execution paths (dynamic analysis)</td>
<td>High</td>
<td>System calls, APIs</td>
<td>Line, path, function</td>
<td>N/A</td>
<td>89</td>
<td>Qsort, md5, matrix</td>
</tr>
<tr>
<td>80</td>
<td>Non-architectural details</td>
<td>Source code instrumentation</td>
<td>Execution paths (dynamic analysis)</td>
<td>High</td>
<td>System calls</td>
<td>Line, path</td>
<td>N/A</td>
<td>90</td>
<td>Text-program, bubble, sky-fire</td>
</tr>
<tr>
<td>84</td>
<td>Architectural details</td>
<td>N/A</td>
<td>Program base cost energy</td>
<td>N/A</td>
<td>Assembly-based instructions</td>
<td>Application</td>
<td>N/A</td>
<td>90</td>
<td>Bubble, heap, quick</td>
</tr>
<tr>
<td>85</td>
<td>Architectural details</td>
<td>N/A</td>
<td>N/A</td>
<td>Assembly-based instructions</td>
<td>Application</td>
<td>ARM7TDI</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>Architectural details</td>
<td>N/A</td>
<td>High</td>
<td>System calls</td>
<td>Execution path, routine</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>86</td>
<td>Architectural details</td>
<td>N/A</td>
<td>N/A</td>
<td>Assembly-based instructions</td>
<td>N/A</td>
<td>i80486</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>87</td>
<td>Architectural details</td>
<td>N/A</td>
<td>Inter-instructions effects (resource-constrained)</td>
<td>High</td>
<td>Assembly-based instructions</td>
<td>N/A</td>
<td>16-bit DSP</td>
<td>N/A</td>
<td>G711, IIR filter, LMS_B</td>
</tr>
<tr>
<td>88</td>
<td>Architectural details</td>
<td>N/A</td>
<td>Inter-instructions effects (circuit state overhead)</td>
<td>N/A</td>
<td>Assembly-based instructions</td>
<td>N/A</td>
<td>ARM7TDI</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>89</td>
<td>Architectural details</td>
<td>N/A</td>
<td>N/A</td>
<td>APIs</td>
<td>Application</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>Non-architectural details</td>
<td>Source code instrumentation</td>
<td>Program base cost energy</td>
<td>N/A</td>
<td>System calls</td>
<td>Application</td>
<td>N/A</td>
<td>N/A</td>
<td>Weather</td>
</tr>
<tr>
<td>74</td>
<td>Architectural level</td>
<td>N/A</td>
<td>Inter-instructions effects (resource-constrained), execution path (dynamic analysis)</td>
<td>High</td>
<td>N/A</td>
<td>Application</td>
<td>Power PC</td>
<td>N/A</td>
<td>Isort, fft, fdct, des</td>
</tr>
<tr>
<td>90</td>
<td>Architectural details</td>
<td>N/A</td>
<td>N/A</td>
<td>Low</td>
<td>Line</td>
<td>DSP</td>
<td>8</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>Non-architectural details</td>
<td>Test programs</td>
<td>N/A</td>
<td>Application, line</td>
<td>N/A</td>
<td>Itachi SH-4 microprocessors</td>
<td>3</td>
<td>Alarm, mutex</td>
<td></td>
</tr>
<tr>
<td>83</td>
<td>Architecture details</td>
<td>N/A</td>
<td>N/A</td>
<td>Assembly-based instructions</td>
<td>Application</td>
<td>Xeon Phi</td>
<td>5</td>
<td>Linpack, scan</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>Non-architectural details</td>
<td>Source code instrumentation</td>
<td>Execution paths (dynamic analysis)</td>
<td>High</td>
<td>System calls, APIs</td>
<td>Line, execution path, function</td>
<td>N/A</td>
<td>90</td>
<td>BBC, sky-fire</td>
</tr>
</tbody>
</table>
operation to memory access pattern such as aligned, stride, and random.

To consider the complex nature of mobile applications, Hao et al.\textsuperscript{38} proposed an estimation framework called eCalc to estimate energy consumption based on source code power profile of the application. For instruction cost model construction, eCalc exploited an instrumented test program to profile power consumption of system calls, APIs, return function statement, and array creation process. In comparison to the work of Khoshbakht et al.\textsuperscript{79} eCalc considered execution path of the application by instrumenting and running a mobile application on a mobile phone using application’s uses cases to mark the execution path. The estimator module of eCalc exploited execution path and instruction power cost profile to estimate application energy consumption. The authors reported that eCalc is accurate up to 91.5\% ground truth value. The proposed framework is lightweight as it does not require any cycle-accurate estimation simulator to profile instruction’s energy consumption. However, path finding method of eCalc is both time-consuming and resource expensive as it has considered dynamic code analysis–based path profiling. Also, eCalc estimates energy consumption of only those applications whose source code is available.

A similar but more detailed study to the work of Hao et al.\textsuperscript{38} has been proposed in one study\textsuperscript{10} to combine per-instruction energy consumption and program analysis for mobile application energy estimation. However, in comparison to eCalc,\textsuperscript{38} the proposed work has considered both path dependent and independent cost functions to improve estimation accuracy. The energy cost of path dependent instructions not only depends on the type of instruction and state of the mobile component but also relies on the weight of an argument such as a number of packets transmitted on wide area network interface and waiting state of the component. Also, the proposed framework annotated the application with energy cost to assist application developers in optimizing mobile application with limited efforts.

In the work of Li et al.\textsuperscript{80} authors combined hardware-based and program analysis to estimate energy demands of a smartphone application’s instructions. Proposed approach in comparison to the work of Hao et al.\textsuperscript{10,38} has considered energy bugs that affect the accuracy of power models when profiling energy consumption of source code instructions (high level). This work too instrumented application and ran it on a mobile phone to capture time-stamped power log in parallel to execution path recording. Regression analysis on logged time-stamped power profile is performed to find per API energy consumption with high accuracy. Based on the execution paths, energy demands of the target application are identified. However, the instrumentation and running application requires a lot of system resources in addition to overhead imposed by instrumentation instructions.

Elen\textsuperscript{s}\textsuperscript{80} is a lightweight energy estimation framework as it does not depend on smart battery sensors for power estimation. The training phase of Elen captures applications execution paths when tested against test-case suites. Workload generator module is responsible for generating load on the mobile phone to test application in a particular execution environment for a test setting. It generates load based on the input of description document and software artifacts modules. The analyzer module considers execution paths and software environment energy profile (SEEP)-based source code energy estimation to assess application energy consumption. The drawback of this scheme is its infeasible assumption that per-instruction power profile for any mobile everywhere is available.

Motivated by the work of Hao et al.\textsuperscript{10,81} Reyhaned et al. proposed EcoDroid,\textsuperscript{82} a system to rank mobile application by estimating application energy consumption across several test cases. EcoDroid estimates application energy consumption using power probe for a set of system calls while automatically generating the test cases for path generations. However, EcoDroid considered power modeling at system call level and overlooked ne granular assembly instruction–based energy estimation that gives more insights into the application’s power consumption behavior.

Jayaseelan et al.\textsuperscript{74} proposed a method to estimate application energy consumption for ARM architecture. In comparison to the works of Hao et al.\textsuperscript{10,38} and Jabbarvand et al.\textsuperscript{82} the proposed study has considered cache analysis of the application too during energy estimation. During application execution, the data and instruction are fetched from the main memory or cache depending on the frequency of memory access. The proposed scheme integrates cache analysis (simulator-based) to the analysis of the application for worst case energy estimation. The proposed scheme lacks in considering the impact of cache hierarchy and lines to cache hit/miss rate.

To profile energy consumption of low-level ARM ISA instructions, Sinha et al.\textsuperscript{76} considered Brutus SA-1100 and a verification design platform to link it with a desktop server (serial link). Moreover, the power supply to ARM core (residing within Brutus SA-1100) is externally provided through variable voltage supplier source. For power measurement, ammeter was used to log current readings. For each instruction, a test program comprising a set of same instances of the target instruction was designed. The proposed scheme concluded that branch instructions are most expensive ones among all instructions. The proposed scheme is accurate as it separates leakage current during the estimation process. However, the cost of the equipment is very high, and also, the proposed work does not consider execution path analysis.

Another approach discussed in the work of Shao et al.\textsuperscript{83} has considered per-instruction energy consumption for “Xeo Phi” processor based on HPC. The proposed scheme considered the effect of the multi-threaded multiprocessing environment to the accuracy of instructions energy consumption. The proposed framework accepts parameters including performance counters, energy performance indicators, instruction, and operand counts, to find the energy of a single instruction. The energy performance indicator is calculated based on the micro-benchmark designed to configure
different cores. The reported estimation accuracy ranges from 1% to 5%. However, collecting performance counter creates dependency on some external tool (i.e., Intel’s VTune tool) to investigate benchmarks to collect required counter variables.

4.2.2 | Simulation-based schemes

Profiling-based methods require external power measurement tool to record time-stamped power profile of an instruction to construct instruction power models. However, simulation-based application’s energy estimation does not require external power measurement tool. Rather, it creates mobile phone’s environment inside a system to measure their behavior on different input datasets. This section briefly describes smartphone application energy estimation literature that considered instruction power modeling based on HPCs simulated over simulators.

Stanley-Marbell et al.84 proposed and implemented a cycle-accurate simulator called Myrmigki to investigate power dissipation rate during instruction execution. The distinguishing features of Myrmigki include DVFS, per cycle architectural configuration, and clock speed setting by modeling CPU cores, off-chip memory, and shared decode cache. The proposed modeled architecture is a 32-bit reduced instruction set computer architecture and is based on 5 stages pipeline. To estimate power consumption, simulator performs a lookup to access the current drop for the memory and CPU operations to calculate the per cycle estimate. However, the proposed simulator is lacking in considering inter-instruction effects during application energy consumption estimation.

Konstntakos et al.85 presented an approach that considered and modeled instruction energy in terms of number of accesses to memory, type of executed instruction, and analog to digital conversion, as described in Equations 4, 5, and 6, respectively. In the mentioned equation, total energy consumed by RAM storage ($E_{\text{RAM}}$) depends on a number of reads and write accesses. Also, $E_{\text{micr} - \text{contl}}$ is attributed in terms of a number of cycles for each instruction. $E_{\text{ADC}}$ is estimated based on number of accesses to memory. The advantage of the proposed model is its flexibility to adjust with any simulator to perform desired operations. However, the proposed work overlooked impact of cache hit/miss on the power performance. Also, it does not consider circuit state overhead of assembly-based instructions.

$$E_{\text{RAM}} = f(\text{ReadAccesses, WriteAccesses, References})$$

$$E_{\text{micr-contr}} = f(\text{Ins 1Cycles, Ins 2Cycles, ..., Ins 5Cycles}),$$

$$E_{\text{ADC}} = f(\text{Accesses}).$$

Another simulator-based approach has been reported in the work of Zhao et al.39 that considered per cycle energy modeling finding instruction’s energy cost to track power consumed by function calls and execution paths. However, the authors overlooked effects of cache hit-rate and miss-ratio to total energy estimation accuracy. Moreover, it overlooked circuit state overhead of the instructions. Alternatively, Brandolese et al.86 modeled instruction’s power cost based on the functional decomposition of tasks that a microprocessor performs at the time of instruction test program execution. Authors constructed power models to breakdown instructions power cost into fetch, decode, and execute activities. Also, authors considered single and multiprocessors to investigate power consumption for each functional unit during application execution.

The technique proposed in the work of Chen et al.87 has considered the format of instructions to estimate energy consumption at processor’s control unit. Moreover, it considered transition signal among coresident instructions to investigate inter-instruction power consumption effects. The proposed framework considered CPU instructions, digital signal processor instructions, and pre-decoder, to identify the instruction(s) and power consumption behavior while placing instruction in different order. The complexity of the proposed technique is very high as it builds energy tables for all control units that takes reasonably larger time. Therefore, accuracy is traded by the estimation speed. The advantage of this technique is its capability to deal with both synthetic and kernel benchmarks.

A similar to the work of Chen et al.87 but detailed simulation-based approach has been discussed in the work of Chang et al.88 to investigate per cycle energy consumption for ARM7 processor. The proposed simulator estimates architectural level energy based on the charge transfer within processor’s circuit, and also, it is robust to noise.

Simulator accurately estimates per-instruction energy to highlight the power-consuming elements within an instruction such as instruction fetch address, op-codes, the number of registers, operand type, and data access. Efficiently optimizing these low-level activities helps to optimize application power use. However, the cache analysis is missing in this study that significantly impacts application performance.

Zhou et al.89 have proposed C-based instruction’s power cost model to estimate application energy consumption. The proposed study improved Embedded strong-ARM energy simulator (EMSIM) to propose iEMSIM that considered improved power models for instruction power consumption as depicted in Equations 7, 8, 9, and 10. Equation 7 demonstrates energy consumption of a single instruction in terms of energy required during fetch, decode, and execute. Alternatively, for a single instruction, Equation 8 computes total fetch energy based on energy consumed while fetching instruction forms memory ($E_{\text{mem}}$) and a total number of CPU cycles ($N_{\text{memcycles}}$) for the fetch state. Equation 9 exploits CPU cycles during which memory was active ($E_{\text{decode}}$) and energy consumed during decoding process (per cycle) to estimate decode stage energy consumption. Energy consumption for executes state is accumulative sum.
of (a) CPU energy in active and an idle state \( (E_{\text{Proc}} \text{ and } E_{\text{Idle}}) \),
(b) accumulated memory energy \( (E_{\text{Memory}}) \), (c) accumulated Universal Asynchronous Receiver/Transmitter (UART) \( (E_{\text{UART}}) \) energy, and (d) energy consumed because of peripheral input/output operations \( (E_{\text{Peri}}) \). The proposed simulator is good for analyzing limited instruction-based operations. However, considering cache analysis, the designed approach is not effective as it overlooked cache analysis support.

\[
E_{\text{total}} = E_{\text{totalfetch}} + E_{\text{totaldecode}} + E_{\text{totaleexecute}},
\]

\[
E_{\text{totalfetch}} = E_{\text{mem}} \times N_{\text{memcycles}},
\]

\[
E_{\text{totaldecode}} = E_{\text{decode}} \times N_{\text{decodecycles}},
\]

\[
E_{\text{totexec}} = E_{\text{Proc}} + E_{\text{Idle}} + E_{\text{Memory}} + E_{\text{UART}} + E_{\text{Peri}}.
\]

In the work of Mehta et al.,\(^{90}\) an instruction power profile is proposed that simulates total energy consumption of instructions based on control paths and arithmetic logic unit. The advantage of the proposed framework is limited simulation speed as it requires a small table lookup during instruction power estimation. The reported accuracy of instruction energy estimation is 8\% for IRSIM-CAP processor. However, the proposed simulator is only usable for the desired processor and cannot be adopted widely.

### 4.3 | Comparison of code analysis–based energy estimation and modeling schemes

This section compares code analysis–based smartphone application energy estimation schemes using a set of parameters such as abstraction level, instruction profiling method, estimation granularity, target processors, program structure analysis, estimation overhead, profiled instruction type, estimation granularity, claimed accuracy, and benchmarks as depicted in Table 1. The following sections briefly debate on each of the categories discussed above to highlight the commonalities and variances among existing schemes.

#### 4.3.1 | Abstraction level

Abstraction level parameter defines what type of information the instruction power models are made from to estimate smartphone energy use. The architectural detail attribute of abstraction level parameter demonstrates that estimation scheme has considered HPCs to construct fine granular power models to estimate energy requirements of a smartphone application. However, finding architectural details is a slow process, and also, it requires offline simulation of required architecture to collect HPCs. Instruction power model is based on CPU cycle to fetch, decode, and execute a target instruction in addition to power consumption during cache hit/miss. Existing state-of-the-art estimation schemes, such as in the works of several authors,\(^{37,74,76,83,84,87,89}\) considered architectural details attribute to construct per-instruction power models. On the other hand, other works\(^{10,76,80,82}\) constructed instruction power models based on profiling through extremal power measuring hardware device. Non-architectural–based instruction power models collect time-stamped power profile for a test program when it is executed on the target processor. However, profiling-based instruction power modeling is costly as they require external power measurement tools such as multimeter, monsoon power meter, and ammeter. Considering architectural-based instruction power modeling, finding per cycle power consumption of embedded processors is a challenging task. The accuracy of non-architectural–based instruction power modeling depends on the resistance of resistor attached to the battery terminals to capture voltage and current drop.

#### 4.3.2 | Instruction profiling method

Attributes of this parameter define the method chosen to estimate per-instruction energy consumption. Attributes of instruction profiling method include test programs and source code instrumentation for instruction power profiling. State-of-the-art smartphone application energy estimation schemes such as in the works of several authors\(^ {10,76,80,82}\) have considered test programs attribute to find per-instruction energy consumption. Alternatively, other works\(^ {10,79,80,82}\) have considered source code instrumentation to find per-instruction energy consumption. However, finding instructions energy based on source code instrumentation method requires high sample rate power capturing tools as low sample rate–based tools can miss various low-level details. Also, it requires strict synchronization between time unit and power profile to eliminate any chance of error. Source code instrumentation-based profiling method is usually preferred to observe energy use behavior of instructions written in high-level languages (ie, system calls and APIs). Test program–based power profiling is suitable for instructions written in a low-level language such as assembly programs. Therefore, in comparison to test program–based profiling, instrumentation method cannot profile power consumption for the activities whose runtime is very small. Moreover, running instrumentation-based application also executes instrumentation code that itself consumes sufficient amount of smartphone battery charge. In comparison, test program's overhead is low, and also, they can handle instructions with relatively small execution time. However, considering power profiling a source code line, source code instrumentation-based profiling is more suitable compared to test program–based profiling.

#### 4.3.3 | Program structure analysis

Considering smartphone applications, the structure of recent smartphone applications is very complex because of
nondeterministic nature of the application. While estimating smartphone application energy consumption, the basic constructs considered during energy estimation by different schemes include (a) program base cost energy, \(^{37,84}\) (b) interdependent behavior of instructions, \(^{37,82,87,88}\) and (c) execution path \(^{10,80,82}\) of the application. Finding base cost energy of the program is simple; however, inter-instruction effect requires checking all the possible combination of the instruction to find the overhead during changing their execution order. Path profiling is a resource expensive process as it runs instrumented application to record the execution path for all possible use cases in offline mode. However, such path profiling is expensive and time-consuming.

4.3.4 Energy estimation overhead

A lightweight estimation framework efficiently uses mobile phone resources while estimating application energy consumption. We have characterized energy estimation overhead of existing energy estimation schemes as low or high based on the architectural design of proposed schemes. Estimation overhead is low when an estimation scheme schedules the resource expensive operations on nearby desktop or cloud servers to save mobile battery charge. In the work of Mehta et al., \(^{80}\) estimation overhead is low as this scheme requires a small look-up table to estimate energy consumption of different activities involved during instruction execution. Alternatively, most of the proposed schemes have not considered their overhead in terms of how much battery power they have consumed during the estimation process. The state-of-the-art schemes such as in the works of several authors\(^{10,39,74,80}\) incur high estimation overhead because of dynamic profiling to mark execution paths of the application for all use cases.

4.3.5 Profiled instruction type and estimation granularity

Profiled instruction type parameter defines the type of the instruction for which time-stamped power profile is captured to estimate energy use of a smartphone application. Existing state-of-the-art smartphone application energy estimation methods have profiled power consumption for (a) assembly instructions, (b) system calls, and (c) APIs. Among all, assembly-based power profile (ie, the works of Konstantakos et al\(^{85}\) and Tiwari et al\(^{17}\)) gives the lowest level of details while considering power consumption from a software perspective. Moreover, system calls and APIs level power profiling (ie, the works of difference authors\(^{10,39,74,80,91}\)) is useful when high-level source code of the application is available to estimate application’s energy consumption. However, assembly instruction power profiling gives more detailed and lower level insights of application power consumption behavior. Considering system calls and APIs, to profile source code of the application is required. However, having assembly power profile, the obj-dump of applications executable can be used to estimate application energy consumption (obj-dump of an application is in assembly).

4.3.6 Target processor, claimed accuracy, and benchmarks

ARM ISA power profile for all architectures is different because of high resource heterogeneity in the underlying architectures. Attributes of the target processor parameter represent the processor model that estimation tool considered during energy consumption estimation for the desired benchmarks. The processor reported in different energy estimation schemes includes Intel 486DX2, itachi SH-4 microprocessors, Power PC, ARM7TDI, and digital signal processor. The application benchmarks considered by various energy estimation schemes include BBC, Sky-fire, Lin-pack, scan, Spec2000 CPU, Mi-Bench, sort, fft, fdecl, and DES. It is noticed that accuracy, as reported by the simulator-based power models, is higher than the rest. However, simulators are comparatively slow and incur high energy estimation time.

5 Smartphone components power modeling and energy estimation

Smartphone energy consumption during a time interval “t” is estimated as an accumulative sum of power consumed by individual components during activity on the mobile phone. Smartphone components power consumption is estimated based on their use level, the number of state transitions, and energy per state transition. Within a mobile phone, components such as Wi-Fi, liquid crystal display (LCD), CPU, and bluetooth are the highest energy-consuming entities. Power consumption by the mobile application is estimated using mobile components power model–based estimation methodology. However, constructing power model for a smartphone component considers external power measurement tools (physical measurement) or self-metering power measurement methodology as highlighted in Figure 4.

Physical measurement–based energy estimation truly depends on the external power measurement tools such as power meter and multimeter, to estimate energy consumption of smartphone applications. Alternatively, self-metering methods exploit built-in smartphone sensors to find energy consumption rate of smartphone applications. The following section classifies state-of-the-art component–based smartphone application energy estimation schemes.

5.1 Taxonomy of components power model–based application energy estimation

Figure 5 presents a thematic taxonomy to classify existing state-of-the-art components power model–based energy estimation schemes. The parameters to classify existing state-of-the-art estimation schemes include power model type, methodology, power measurement source, granularity, training mode, and power modeling approach.
• Power model type parameter describes that whether existing energy estimation schemes have followed external power measurement or self-metering–based methods to construct power models. External power measurement methods depend on power estimation tools for obtaining voltage and current drop across the sense resistor. Alternatively, self-metering estimates smartphone applications energy use based on SOC estimation methods.

• Methodology defines the power modeling approach based on the kind of input variables the model uses. Utilization-based methodology co-relates smartphone component power use to the resource use. The utilization-based method exploits HPCs to model power consumption of smartphone subcomponents such as CPU fetch cycles, cache hit rate, pipeline stalls, and a number of cache misses.

• Power log source describes whether a sense resistor is used to log power measurements during the experiments or fuel gauge–based voltage/current sensors are used. Sense resistor represents a small resistance resistor that is placed on the power rail of the target mobile component for measurements. Alternatively, fuel gauge–based sensors are built-in sensors usually accessed through android-based APIs.

• Granularity defines the level to which estimation scheme proposes a method for energy estimation. In the literature reported in this paper, authors proposed estimation methodologies to forecast energy consumption at software level or hardware level. Also, some of the estimation methods (power tutor) estimate energy at both application and system level.

• Training mode locality describes the location where mobile component power training is performed to construct power models. Mobile component power training is carried either at smartphone device or on desktop/cloud servers. In the case of on-device training, a sufficient amount of energy is required to train all the mobile components that put extra overhead on the mobile device and deplete mobile battery charge quickly. On the other hand, off-smartphone training minimizes burden on the mobile device by scheduling training on an external device to augment device battery lifetime.

• Attributes of support parameter define the basic user of the proposed estimation tool. The users are classified as application developers and testers.

5.2 State-of-the-art smartphone components power model–based energy estimation and modeling schemes

Jiang et al\textsuperscript{92} proposed a study to investigate and compare the power consumption of 3 input modalities including soft key, speech to text, and swipe. Authors performed primary and secondary experiments to investigate these 3 modules with and without user-based context information. To capture voltage and current drop, the shunt resistor is attached to the battery terminal to capture readings during activity on the mobile phone. It was concluded that for short size messages (14-30 characters), soft texting is most energy-efficient. However, for long messages, soft key outperforms than others too in terms of power consumption. Also, power consumption for different applications such as email via a web browser, video streaming, online games, news/weather, and email via APP has been investigated with non-textual online/offline activities. This study helped to find out how battery life of the mobile phone can be increased. However, apart from the input modalities, numerous other factors such as bugs in the code, Wi-Fi locks, and an aging factor of mobile batteries also affect mobile battery power consumption that is overlooked in this study while classifying 3 input modalities.

Interaction with a mobile phone to adjust settings during an experiment impacts estimation accuracy because of a sudden rise in LCD power use. To minimize interaction with a mobile phone, Rice et al\textsuperscript{19} proposed a framework that
downloads (online mode) scripts to generate annotated traces for mobile application energy consumption assessments. Proposed framework consists of a desktop server (time-stamped power recording) and power measurement tool (power capturing). Mobile-based module runs scripts on a mobile phone and uses power measurement tool to record and transfer traces to the remote server. To assure the strict synchronization between mobile and remote cloud, the framework uses a test client application that formulates synchronization pulse. During the offline phase, power profile is processed (regression analysis) to construct power models. Authors considered a network study to examine the power consumption of 2G/3G radios. It was concluded that 2G is more power-consuming than 3G. The advantage of proposed framework is its versatility to support noninvasive tasks, batch job processing, automated test generation, and execution.

Rice et al.\textsuperscript{35} considered hardware instrumentation method to investigate power draw rate of smartphone application(s). The key modules of the proposed framework include hardware equipment and a software stub. Similar to the work of Rice et al.\textsuperscript{19} this scheme too opted automated test generation to acquire test scripts from the remote server to capture and transfer power log of application to remote server. This study too focused on network study to examine the power consumption of 2G/3G radios. It was concluded that 2G is more power-consuming than 3G. The advantage of proposed framework is its versatility to support noninvasive tasks, batch job processing, automated test generation, and execution.

FIGURE 5 Taxonomy of mobile component power model–based energy estimation. API indicates application program interface; CPU, central processing unit; LCD, liquid-crystal display

identifying power-consuming processes within a mobile phone. Power memo\textsuperscript{93} proposes a novel approach to attach process ID on a socket to identify the source of packets. The design of power memo consists of 2 modules one running on host side where other at target side. A host side module accesses the data acquisition (DAQ) and emulates the mobility pattern to assess network communication power consumption. Whereas, target side module exploits k-probes and u-probes to acquire the power log as it runs inside the kernel module. The factors including radio type, the location of the access point(s), peak signal-to-noise ratio, and network bit rate significantly impact input/output estimation accuracy and have been overlooked in this study.

Web browser power estimation helps to identify the power consumption rate of a web browser during data transfer and connection establishment phases. Thilagarajan et al.\textsuperscript{94} proposed a web browser power inspection tool that consists of mobile phone, power tracker, and a server module to estimate energy consumption. The estimation approach considered web page loading time, network data rates, peak signal-to-noise ratio, signal strength, and web page caching behavior to estimate power consumption for a set of web pages. The proposed approach is efficient as it considers multiple parameters to estimate energy consumption. However, it has only considered a few web pages to estimate energy consumption. However, the interest of the smartphone application developers is on estimating energy needs for a complete web session that is overlooked by this study.

Application and system developers rely on energy estimation information for smartphone applications to optimize the code for augmenting device lifetime. AppScope\textsuperscript{95} is an online android-based energy estimation tool that resides within OS kernel module to estimate energy consumption. It used event-driven monitoring method for accurate
energy estimation. For energy estimation of smartphone applications, AppScope depends on system call traces and android binder inter-process communication. The design of AppScope consists of (a) processes identifier acceding hardware components, (b) use statistics analyzer, and (c) estimator to estimate the energy of application based on collected use statistics and states of hardware components. The main limitation of AppScope is its linear component power model that cannot capture the power state of the hardware components (Wi-Fi). Similar to AppScope, Jung et al and Pathak et al too have used system call traces to estimate power consumption of smartphone components.

Power-prof exploited genetic algorithms to propose power models for smartphone components to represent their dynamic power consumption behavior. Power-prof considered smart battery interface (gauge sensors) for capturing current and drop through android built-in power APIs. Power model for each mobile component considered 4 power states to represent power consumption behavior with limited search space to speed up power estimation process. Moreover, to optimize mobile battery consumption, power-prof offers offline training mode to perform resource expensive execution there. However, during the training phase, it overlooks inter-mobile components dependency that affects estimation accuracy. Also, to improve the estimation accuracy, power-prof can be tuned to widen the search space to integrate more power details in existing model. On-device power model generation does not require external hardware devices to construct mobile phone components power models. The proposed framework as discussed in the work of Zhang et al called Power Booter consists of 2 components including training and model construction. Training phase profiles power consumption of each mobile component using built-in smart battery interfaces, and model construction module applies linear regression to construct power models. However, a training phase is resource expensive and time-consuming. Power Booter exploits voltage discharge curve that varies with time and age of the battery. However, the advantage of this method is its adaptability as it does not depend on external hardware device for power modeling.

S. Dong et al presented Se-same, an energy estimation framework, to propose a self-modeling scheme to construct high-rate and accurate smartphone components power models. Se-same consists of a collector, model modeling, and model constructor modules, to estimate application energy consumption. Among all modules, the collector being located within the kernel collects power and use logs based on Advanced Configuration and Power Interface. Alternatively, model modeling generates high-rate mobile components power model based on prediction transformation component. Estimation overhead is very limited as it exploits external physical server to execute resource expensive tasks. The acceptability of Se-same is very low as it only considered those components that are visible to OS for mobile application energy estimation.

Se-same requires OSs kernel instrumentation to capture system power and use level statistics. Smart energy monitoring system (SEMO) exploits inspector, recorder, and analyzer modules to estimate application energy consumption. Among all, the inspector is responsible for capturing health, voltage scale, and current battery capacity of the mobile phone to intimate user about battery health before it reaches to critical condition. The recorder collects numerous batteries and program relevant attributes, including execution time, power consumption, and applications running on the system. The analyzer module exploits recorders profile in addition to power use history of application to the forecasting energy consumption of the mobile application. This framework overlooks the impact of mobile components that are invisible to the OS.

Fine granular Eprof has opted last-trigger accounting strategy to suppress the asynchronous power dissipation effects of smartphone components. During power profiling process, Eprof considered various types of energy bugs, ie, tail energy and inconsistent power state wake locks. Eprof is widely adaptable as it estimates energy at both fine and coarse granular granularity. Nevertheless, the proposed scheme ignored the amount of energy consumed by the OS policies and underprivileged software designs.

Table 2 compares aforementioned energy estimation schemes for smartphone applications based on the parameters as presented in thematic taxonomy.

5.3 | Comparison of components power model–based energy estimation and modeling schemes

This section compares state-of-the-art component–based smartphone application energy estimation schemes to highlight commonalities and variances among existing schemes. The parameters to compare existing schemes include power model type, methodology, power log source, granularity, training mode locality, and objective function as shown in Table 2. Given below is a detailed comprehensive debate on each of these categories.

5.3.1 | Power model type

Attributes of power model type represent whether estimation scheme has considered external physical measurement or self-measurement–based modeling paradigm. Existing schemes, such as Device Under Test (DUT), Rice and Hay, Netw, Power Memo, Browser, and Skype considered external measurement-based power estimation model. Alternatively, state-of-the-art schemes, such as Power-prof, Power Booter, Se-same, P-top, Eprof, and SEMO have chosen self-measurement–based energy estimation model. External measurement-based estimation models are more accurate compared to self-metering–based energy estimation because of error-prone nature of fuel
TABLE 2 Comparisons of components power model–based energy estimation and modeling schemes

<table>
<thead>
<tr>
<th>Reference</th>
<th>Power Model Type</th>
<th>Methodology</th>
<th>Power Log Source</th>
<th>Granularity</th>
<th>Training Mode Locality</th>
<th>Objective Function</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>92</td>
<td>External power measurement</td>
<td>Utilization-based</td>
<td>Sense resistor</td>
<td>App</td>
<td>Off-device</td>
<td>Characterize power consumption of 3 input modes</td>
<td>Tester</td>
</tr>
<tr>
<td>19</td>
<td>External power measurement</td>
<td>Trace-driven</td>
<td>Sense resistor</td>
<td>App</td>
<td>N/A</td>
<td>Automated energy profiling</td>
<td>Tester</td>
</tr>
<tr>
<td>35</td>
<td>External power measurement</td>
<td>Trace-driven</td>
<td>Sense resistor</td>
<td>App</td>
<td>N/A</td>
<td>Automated energy profiling</td>
<td>N/A</td>
</tr>
<tr>
<td>93</td>
<td>External power measurement</td>
<td>Trace-driven</td>
<td>V/I clamps</td>
<td>App</td>
<td>Off-smartphone</td>
<td>Asynchronous behavior of Wi-Fi power consumption</td>
<td>Tester</td>
</tr>
<tr>
<td>94</td>
<td>External power measurement</td>
<td>N/A</td>
<td>Sense resistor</td>
<td>App</td>
<td>Off-smartphone</td>
<td>Web component rendering energy</td>
<td>Developers</td>
</tr>
<tr>
<td>95</td>
<td>Self-measurements</td>
<td>Trace-driven</td>
<td>N/A</td>
<td>App</td>
<td>On-smartphone</td>
<td>System call rendering</td>
<td>Tester and developers</td>
</tr>
<tr>
<td>97</td>
<td>Self-measurements</td>
<td>Trace-driven</td>
<td>Fuel gauge</td>
<td>App</td>
<td>Off-smartphone</td>
<td>Unsupervised power profiling</td>
<td>Developers</td>
</tr>
<tr>
<td>98</td>
<td>Self-measurements</td>
<td>Utilization-based</td>
<td>Fuel gauge</td>
<td>App</td>
<td>On-smartphone</td>
<td>Online power model generation</td>
<td>Tester</td>
</tr>
<tr>
<td>99</td>
<td>Self-measurements</td>
<td>Trace-driven</td>
<td>Fuel gauge</td>
<td>App</td>
<td>Off-smartphone</td>
<td>Self-power model construction</td>
<td>N/A</td>
</tr>
<tr>
<td>100</td>
<td>Self-measurements</td>
<td>N/A</td>
<td>N/A</td>
<td>App</td>
<td>N/A</td>
<td>Smart energy monitoring</td>
<td>N/A</td>
</tr>
<tr>
<td>101</td>
<td>Self-measurements</td>
<td>Trace-driven</td>
<td>Fuel gauge</td>
<td>App</td>
<td>Off-smartphone</td>
<td>Energy bug finding</td>
<td>Tester</td>
</tr>
</tbody>
</table>

gauges voltage and current sensors. However, external measurement-based estimation requires lab-setting environment to conduct experiments. The accuracy of external measurement-based modeling highly depends on (a) sample rate of power monitoring devices such as a multimeter, power meter, and ammeter, (b) device disruption during the training phase, (c) and calibration of the device under test. Existing energy estimation schemes considered various types of equipment to capture voltage and current drop during experiments such as PCI-6229 DAQ, PCI-MIO-16E-4, PCI-6115 DAQ, and PCI-MIO-16E-4 to predict smartphone/application energy consumption. The accuracy of self-metering–based energy estimation schemes depends on (a) error rate of smart battery’s voltage/current sensors, (b) fuel gauges sensor updating rate, and (c) mobile power API’s access rate. Self-metering–based estimation solutions do not require extra external hardware equipment to capture power readings as they use API to access voltage and current sensors.

5.3.3 | Power log source

Power log source parameter describes the hardware equipment opted as a source to collect current and voltage drop readings. Existing state-of-the-art energy estimation schemes have either considered sense resistor or fuel gauge hardware sensors to access current and voltage drop. Estimation schemes, such as DUT,34 Rice and Hay,19 Netw,35 Power Memo,93 and Skype,92 have considered a sense resistor to collect current and voltage drop during activity on mobile phone to construct power models. Alternatively, Power Booter,98 Power-prof,97 Se-same,99 Eprof,101 and P-top102 have exploited fuel gauge sensors to log voltage and current drop using built-in voltage/current sensors. Sense resistor–based methodology uses an extra external resistor placed at mobile component's power supply rail to monitor voltage/current drop. The accuracy of this category of estimation schemes highly depends on the resistance of the sense resistor. Sense resistor–based methodology is helpful when evaluating power consumption behavior of single mobile component; however, they are not accurate while monitoring the behavior of an application in terms of mobile components power use. This is because collectively instrumenting every component of the mobile phone for application energy estimation increases the complexity of circuits and total impedance. Alternatively, fuel gauge–based estimation does not require external hardware resources and uses built-in OS's
power management APIs to collect voltage/current drop. However, fuel gauge–based methods do not directly estimate per-mobile component energy. For a single mobile component power capturing, it switches off all the mobile components except one under analysis. However, some of the components reside on the same physical hardware and hinder fuel gauge–based estimation method to measure current/voltage accurately (ie, Bluetooth and Wi-Fi).

5.3.4 | Granularity

Attributes of granularity parameter describe the extent at which estimation scheme estimates energy consumption. Estimation schemes facilitate developers to estimate energy consumption either at the application level or at the system level. Considering application level estimation, existing schemes offered energy estimation at diversified granularity levels such as source code line, assembly instruction, path, application, function, and thread. Among all granularity levels, fine granular level estimation is more accurate. However, fine granularity level estimation (mobile component, source code line, and path) is resource expensive and also requires extensive profiling of target application/component for a long period of time\(^{34,97,101}\). Alternatively, coarse granularity–based estimation\(^{19,35,92,102}\) requires (a) few system resources, (b) limited energy profiling time, and (c) low resource monitoring. In comparison to software level granularity, estimating energy consumption at mobile component levels such as Wi-Fi, LCD, radio, and CPU (DUT\(^34\)) requires (a) extensive profiling, (b) per-component power profile isolation, and (c) offline power traces analysis (regression analysis) to extract coefficient of power models.

5.3.5 | Training mode locality

The training phase is the most resource rigorous process during mobile application power modeling. In this phase, it extracts coefficient of mobile phone power models. Training phases are performed either on mobile device or off-smartphone. On-device training incurs high estimation overhead and quickly depletes mobile battery charge. Alternatively, off-device training augments device battery lifetime as it schedules power model’s coefficient finding process on the remote cloud or desktop systems. However, the drawback of off-smartphone-based computation is the high dependency on the external hardware. Existing state-of-the-art energy estimation schemes such as in the works of several authors\(^{34,92,93,97,99,101}\) has scheduled resource-intensive tasks on the desktop servers to reduce estimation overhead. Alternatively, Power Booter\(^98\) has not relied on the external hardware for the processing of resource expensive tasks.

5.3.6 | Objective function

Objective function defines the core motivation for each of mobile application energy estimation scheme. State-of-the-art energy estimation schemes have targeted (a) automatic power model generation, (b) low overhead–based estimation, (c) asynchronous network component behavior monitoring, (d) digging loopholes within application, (e) self-power model construction, (f) web component rendering cost estimation, and (g) investigating the resource expensive activity within mobile phone among talk, text, and swipe operations.

Table 3 compares state-of-the-art energy estimation schemes based on their findings. Among all estimation schemes, a set of approaches have estimated mobile application energy whereas others estimated energy for mobile components, mobile phone handling, and voice conversation. Moreover, it showed a set of benchmarks that are considered by the energy estimation schemes during experimentation. For instance, DUT\(^34\) reported that during the run of Equake on HTC dream mobile phone, Bluetooth consumed 44.9 mW power. Similarly, Netw\(^35\) states that during an idle

<table>
<thead>
<tr>
<th>Reference</th>
<th>Citations</th>
<th>Benchmark(s)</th>
<th>Key Finding(s)</th>
<th>Accuracy, %</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>888</td>
<td>Equake, vpr, gzip, mcf</td>
<td>Component energy: Bluetooth power = 44.9 mW</td>
<td>N/A</td>
</tr>
<tr>
<td>19</td>
<td>84</td>
<td>TCPdump, libpcap</td>
<td>Network energy: 3G idle = 0.6 W, 2G idle = 0.7 W, Wi-Fi idle = 0.9 W</td>
<td>N/A</td>
</tr>
<tr>
<td>35</td>
<td>66</td>
<td>TCPdump</td>
<td>Mobile energy: Mobile idle full-brightness = 1-1.5 W, Mobile idle dim-light = 0.3-0.7 W, Standby = 0.1-0.4 W</td>
<td>N/A</td>
</tr>
<tr>
<td>93</td>
<td>2</td>
<td>N/A</td>
<td>Apps energy: Unmodified JPEG = 23.3 J, modified JPEG = 15.9 J</td>
<td>N/A</td>
</tr>
<tr>
<td>94</td>
<td>120</td>
<td>Aol, picasa, live.com</td>
<td>Network energy: 3G upload energy = 25 J for 26 KB, 3G download energy = 15 J for 26 KB</td>
<td>N/A</td>
</tr>
<tr>
<td>72</td>
<td>0</td>
<td>Oracle-modality</td>
<td>Apps energy: Apple rendering = 455 J, Wikipedia = 35 J, Picasa = 15 J, Yahoo = 18.5 J</td>
<td>91</td>
</tr>
<tr>
<td>97</td>
<td>29</td>
<td>N/A</td>
<td>Text/voice energy: Swype energy = 0.57 J/character, Type energy = 0.11 J/character, Skype energy = 0.45 J/character</td>
<td>97</td>
</tr>
<tr>
<td>98</td>
<td>706</td>
<td>Synthetic benchmark</td>
<td>0.145 Error for 95% percentile</td>
<td>80</td>
</tr>
<tr>
<td>99</td>
<td>3</td>
<td>JBenchmark, 3DMarkMobile</td>
<td>N/A</td>
<td>95</td>
</tr>
<tr>
<td>100</td>
<td>33</td>
<td>N/A</td>
<td>95% accuracy = 1 estimation/s, 88% accuracy = 1 estimation/10 ms</td>
<td>N/A</td>
</tr>
<tr>
<td>101</td>
<td>11</td>
<td>FFT, filterbank, matrixmult</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>102</td>
<td>68</td>
<td>Synthetic benchmark</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
state of a mobile phone with full brightness level (LCD), magic phone consumes 1 to 1.5 W power.

6  |  ENERGY EFFICIENCY AND TECHNOLOGY ADVANCEMENTS

Because of high functionality demands of users, the advancements in IT infrastructure are rapidly growing. With advancements in IT infrastructure, the rising trend in energy needs has become an integral issue for existing IT solutions. The high demands of users call for on-demand data processing, local edge–based processing, and seamless application execution.

Cloud computing offers on-demand application execution environment to offer services based on pay-as-you-go service model. Within the cloud, 20% of the energy is saved using server and network resources. The cooling equipment consumes 60% to 70% of the energy budget of a cloud data center. Hardware- and software-based techniques help to minimize total energy of a data center. The techniques such as energy-aware scheduling, server consolidation, and DVFS optimize energy consumption within servers and network communication. Considering network communication, internet traffic significantly impacts the total energy consumption and is minimized based on considering the effect on energy consumption while replicating data for high availability.

Conceptually, Internet of Thing (IoT) consists of a set of entities that are (a) identifiable, (b) communicating, and (c) interactive. Entities within IoT are operated with small size batteries. In IoT, the major portion of the energy of an IoT entity is consumed while interacting with entities in its surroundings. The energy-efficient techniques consider piezoelectric materials, thermoelectric, or micro solar panels to harvest energy. Moreover, the notion of distributing computations to reduce the communication cost too increases the lifetime of IoT sensors. The main element to reduce energy in IoT includes low energy computing architectures, Radio-Frequency Identification (RFID) communication mode optimization, and near-field communication. Among various application areas of IoTs, smart health, smart Home, smart retail, smart grid, smart agriculture, smart environment monitoring, and smart logistics, are a few to mention. A few example applications of smart health include sleep monitoring and fall monitoring of the patients.

7  |  OPEN RESEARCH ISSUES AND CHALLENGES

This section presents several research issues in this domain of research. Handling these research issues will optimize mobile application energy estimation for effective resource use to augment device battery lifetime.

7.1  |  Low estimation accuracy vs battery aging factors

State-of-the-art smartphone application energy estimation schemes forecast energy consumption based on SOC estimation while considering the ratio of current battery charge capacity to the normal capacity. Normal capacity represents the total storage capacity of the battery usually defined by the manufacturer of the mobile phone battery. However, SOC estimation accuracy is limited because of the architectural flaws in their construction. For instance, for each mobile phone model, the SOC estimation error is varying. State of charge estimation methods, ie, Rint model and counting model, do not truly represent the accurate capacity of the battery. This is because these models do not consider the impact of aging, operational history, and size of the battery, during SOC estimation process. Historically, it is proven that for lithium-based batteries, discharge rate varies with the same battery use (known as a lithium-ion aging factor).

As a result, state-of-the-art SOC estimation methods report voltage drop that is higher than its true value. Therefore, power models generated for one type of mobile or battery cannot be generalized for every model of batteries. For mobile phone batteries, Rint model–based SOC estimation gives 11% estimation error. Also, counting-based method incurs 5% to 2% error. Another factor that affects estimation accuracy is the high difference between smartphone's battery interface updating rate and OS's API access rate. For instance, the highest updating rate of smart battery interface is 4 Hz, whereas, Linux OS updates the P-state residency for CPU component at 250 Hz rate. The rate of the smart battery interface affects the power model generation time.

Existing energy estimation schemes that uses built-in smart battery interfaces for energy estimation does not accommodate the error generated by SOC while reporting mobile application energy estimation.

7.2  |  Smartphone resource limitations and bad smells

Nowadays, mobile phones are heavily used portable devices. It is estimated that during 2015, android application development marked has reached to US $38 million because of exponential growth in mobile phone users. However, still, mobile phone use is limited because of their limited resources. Usually, mobile phones are equipped with limited battery capacity, limited storage, small size screen, low capacity processors, and highly vulnerable circuits. The average battery lifetime of mobile phones is very limited because of resource-hungry nature of mobile applications. For instance, a context-aware resource-rich mobile application, ie, proximity-based applications, enriches user experience at the cost of quick mobile battery charge depletion. The execution speed of mobile phones is limited because of (a) limited CPU clock rate(s), (b) small cache size to host frequently used data, and (c) limited RAM storage capacity. Moreover, mobile device's visualization quality is very poor because of the small size of the screen. Considering security perspective, mobile devices are vulnerable compared to
desktop servers. This is because mobile devices offer low-resistance against sensitive vulnerable attacks to save battery charge. Also, high negligence of users during mobile phone makes it vulnerable to attacks. In addition to vulnerability issues, mobile applications sometimes abnormally use mobile’s battery charge because of energy bugs embedded within applications or hardware. Hardware energy bugs are difficult to track and mainly occur because of (a) faulty batteries, (b) damaged mobile battery chargers, (c) infected memory cards, and (d) damaged SIM cards. For software applications, within an OS changing OS configuration impacts the mobile battery power consumption rate. For instance, incorrect SetCPU configuration for kernels overclocking leads to a sudden rise in mobile battery-power-use rate. Similarly, infected mobile application (bad smells) and frameworks are difficult to track as energy bugs do not affect the functioning of the application. A no sleep bug hinders a mobile phone component to go into a sleep state that as a result, depletes mobile battery charge. A mobile application, with no sleep bugs, acquires a lock on a mobile component and does not release it for a long period of time. Similarly, sleep conflicts bad smell due to a situation in which an application acquires a lock on a mobile component and then CPU moves to sleep state without waiting for the application to release the component.

7.3 | Smartphone application nondeterministic behavior
This part debates on the issues relating to the nondeterministic nature of smartphone applications.80,85

7.3.1 | Execution path estimation
The battery use of a smartphone application profoundly depends on the path it selects at runtime. During runtime, application execution path depends on the current input, use case of the application, and applications historic data. Dynamic application path profiling instruments application to identify the execution paths during application execution on a server (offline mode) while opting different test cases.10,39,74,80 The energy estimation of the program is estimated based on the energy cost model and execution path marked using dynamic path profiling. However, the instrumentation and profiling phases are time-consuming and resource expensive. Also, instrumentation requires source code of the application.

7.3.2 | Loop bound estimation
Within mobile application's source code, some portion repeatedly executes for a fixed time. Finding a solution to identify and estimate iteration of repetitive portion of code can augment estimation methodology while considering static code analysis–based solutions.128–130 The factor that affects loop bound estimation includes (a) initialization, (b) termination, and (c) growth rate of the variable. However, these 3 parameters are not known always depending on the structure and need of the program. Therefore, there should be some method that should estimate loops within the assembly-based application to find the repeated portion of code.

7.3.3 | Storage access estimation
Nowadays, smartphones are equipped with the multilevel cache to speed up the application execution time by directly accessing data and instructions from the cache hosted locally. Simulators are used to track memory access pattern for mobile application.131 However, simulation-based solutions are offline and computationally very slow. Rather than opting dynamic code analysis, static code analysis method can be used to estimate memory access pattern of mobile application’s code. Also, the instruction power models for ARM ISA for both cache and memory model too will be different because of dissimilar access time.

7.4 | Architectural incompatibility and high estimation overhead
The instruction power profile for a set of instructions using heterogeneous mobile architectures is dissimilar because of their resource objectives. For instance, ARM7 is high performance whereas ARM15 is energy-efficient architecture. Estimation overhead is one of the most important issues in code-based estimation domain. Dynamic profiling runs the application to analyze source program. However, to analyze the program, the mobile application is instrumented that requires annotating the application. Also, execution of annotation instructions itself consumes a sufficient amount of energy. Dynamic profiling increases estimation time and uses a lot of mobile energy.

8 | DISCUSSION
The recent trend to execute tasks on smartphone calls for optimizing legacy applications for effective battery resource use. Smartphone application energy estimation creates an opportunity for developers to reconsider their application design at earlier development stages for effective battery resource use. A mobile application energy estimation method uses either mobile component–based power model or code analysis–based estimation to forecast application energy consumption. A mobile component–based power estimation method exploits external hardware equipment, ie, power meter, multimeter, or smart battery interfaces, to collect power traces to construct power models. Component-based power models are not highly accurate as they use SOC estimation to monitor mobile application power consumption. However, because of the architectural flaws in mobile battery construction, SOC estimation methodologies do not report a true capacity of mobile battery charge. On the other hand, hardware-based mobile application energy estimation solutions are time-consuming and resource expensive because of (a) inter-mobile
component dependency, (c) mobile component’s wake-locks, and (d) energy bugs within mobile applications. Considering software aspects of mobile applications, code analysis assists in estimating power consumption based on power cost model of instructions within application’s source code.

Code-based energy estimation helps application developers to estimate application energy consumption at diversified granularity levels such as path, line, application, and function level. Mobile application energy estimation is based on power cost models for either high-level source code instructions or low granular assembly instruction. High-level source code power cost models require original source code of the application to estimate application energy consumption. However, assembly instructions power profile-based estimation uses assembly code of the application for energy estimation (extracted using obj-dump of the executable file). ARM ISA power profile is created using external hardware-based power capturing tools (profile-based) or using cycle-accurate simulators. Compared to profiling-based power modeling, cycle-accurate simulators are extremely slow. Based on the ARM ISA power profile and program analysis mobile application, energy consumption is estimated. However, program analysis faces various challenges because of nondeterministic behavior of today’s mobile application. Current estimation methods exploit dynamic code analysis of application to find the execution paths to estimate application energy consumption. However, dynamic code analysis requires annotating source code of the application. Annotated code itself consumes a significant amount of energy to record execution path of the application when executed on a mobile phone. Also, during dynamic code analysis, a sufficient amount of energy is consumed that depletes mobile battery charge. As a result, estimation time and overall estimation overhead surge when considering dynamic code analysis-based mobile application energy estimation. In addition, for recent mobile phone, the inclusion of multilevel cache too increases the opportunity to accurately estimate smartphone application energy consumption rate by identifying the storage location of source code within an application.

9 | CONCLUSION AND FUTURE WORKS

Smartphone application energy modeling and estimation is an emerging area of research because of its long lasting applications in various resource critical domains such as (a) mobile cloud computing, (b) mobile code optimization, (c) energy bug detection, and (d) hardware performance monitoring. This study has extensively explored smartphone application energy modeling and estimation schemes to critically analyze existing schemes for highlighting their advantages and limitations. It has proposed thematic taxonomies to classify existing literature into several categories. Based on the thematic taxonomies, it has compared existing schemes to highlight the commonalities and dissimilarities among modeling and estimation schemes. Finally, it has presented several open research issues and challenges that need further research to propose lightweight accurate energy estimation frameworks.

For mobile components power model-based estimation, energy consumption estimation accuracy highly depends on (a) fuel gauge updating-rate, (b) OS’s API power capturing-rate, (c) SOC estimation error, (d) measurement tools accuracy and sample-rate, and (e) interdependency among smartphone components. Alternatively, the accuracy of code analysis-based smartphone application energy estimation depends on (a) ARM ISA power profiling accuracy, (b) execution path estimation, (c) cache analysis of source code, and (d) accurate loop bound estimation. The design of energy estimation scheme should be lightweight ideally to be able to increase smartphone’s battery lifetime. In the future, we plan to propose a lightweight energy estimation framework based on static code analysis of mobile application to precisely predict execution paths and storage location of data.

ACKNOWLEDGMENTS

This work has been partially supported by the Malaysian Ministry of Education under the High Impact Research Grant of University Malaya UM.C/625/1/HIR/MOE/FCSIT/03, by the Government of Russian Federation, grant 074-U01, by the Visiting Professor Program at King Saud University, and by Finep, with resources from Funtell, grant no. 01.14.0231.00, under the Radiocommunication Reference Center (Centro de Referência em Radiocomunicações—CRR) project of the National Institute of Telecommunications (Instituto Nacional de Telecomunicações—Inatel), Brazil.

REFERENCES


