International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 03 Issue: 04 April 2025
Page No: 1625-1633
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2025.0231

A Survey on Energy Leak Detection in Android Applications: Approaches and Challenges

Mrs. Jayalakshmi [1], Dr.R. G Suresh Kumar[2], A Divya Priya[3], A Dheepika[4], D Jenifer[5]
[1]Assistant professor, Dept of CSE, Rajiv Gandhi College of Engineering and Technology, Puducherry, India
[2]Head of the department, Dept of CSE, Rajiv Gandhi College of Engineering and Technology, Puducherry, India.
[3,4,5] UG Scholar, Dept of CSE, Rajiv Gandhi College of Engineering and Technology, Puducherry, India.
Email ID: *jayalakshmi_r@rgcet.edu.in[1], sureshkumar_rg@rgcet.edu.in[2], divyapriyaanbazhagan@gmail.com[3], dheepikaanbu01@gmail.com[4], Jeniferjenifer1205@gmail.com[5]*

Abstract
With the rapid growth in mobile application usage, energy efficiency in Android applications has become a critical research focus. Excessive battery consumption, often due to energy leaks such as mismanaged wake-locks and other resource inefficiencies, degrades user experience and device longevity. This survey reviews current methodologies for detecting energy leaks in Android applications, including static analysis, dynamic analysis, hybrid analysis, machine learning approaches, and testing frameworks for identifying energy-related code smells. Static analysis examines code without execution to detect potential energy inefficiencies, while dynamic analysis observes app behavior during runtime to identify actual energy drains. Hybrid analysis combines both approaches, enhancing detection accuracy. Recently, machine learning models have been applied to analyze app performance data, shifting the focus from traditional testing to predictive diagnostics. This survey highlights key advancements, challenges, and emerging trends in energy leak detection and advocates for integrating machine learning algorithms. By leveraging app usage data, machine learning offers a scalable, accurate, and proactive solution to energy inefficiency in Android applications, paving the way for more sustainable mobile development.

Keywords: *Android applications; Dynamic analysis; Machine learning; Static analysis.*

1. Introduction
With the proliferation of mobile applications and the widespread adoption of smartphones, energy efficiency has become a key area of research. The study analyzed 32 popular apps across 16 categories, revealing energy consumption trends. A dataset of 14,064 user reviews was collected, with 8,007 specifically addressing energy concerns, highlighting significant user dissatisfaction. All analyzed apps exhibited energy consumption issues, impacting user ratings and adoption. Data was gathered over five years to ensure relevance. Android applications, in particular, often experience energy inefficiencies due to improper management of power resources, leading to excessive battery drain and a diminished user experience. These inefficiencies frequently arise from mismanaged wake locks, unbalanced power control calls, and complex, event-driven app behaviors that are challenging to predict and control (Pathak, A. et al.,2012; Abbasai, A. et al.,2018; Sachin C,2024). Energy leaks occur due to inefficiencies in software that lead to unnecessary energy consumption, particularly in mobile applications where battery life is a critical concern. Wake lock and resource leaks are common Android issues that cause excessive energy consumption. Wake lock leaks occur when a wake lock is acquired but not properly released, keeping the device in a high-power state. Resource leaks happen when apps

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 03 Issue: 04 April 2025
Page No: 1625-1633
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2025.0231

allocate resources but fail to release them, leading to unnecessary energy usage. Both types of leaks are often attributed to inexperience among developers and a lack of adequate guidelines or tools to identify energy inefficiencies, making it crucial for developers to implement best practices in resource management to enhance energy efficiency in their applications. These leaks can occur due to various programming flaws, often categorized as "code smells." Code smells are indicators of poor design or implementation choices that can negatively impact the performance and energy efficiency of an application (Palomba, F et al.,2019; Khan, M. U et al., 2020).

```
private void updateWakeLock()
   {
      if( this.mLoggingState == Constants.LOGGING )
      {
         PreferenceManager.getDefaultSharedPreferences(
this ).registerOnSharedPreferenceChangeListener(
mSharedPreferenceChangeListener );

         PowerManager pm = (PowerManager)
this.getSystemService( Context.POWER_SERVICE );
         this.mWakeLock = pm.newWakeLock(
PowerManager.PARTIAL_WAKE_LOCK, TAG );
         this.mWakeLock.acquire();
      }
      else
      {
         if( this.mWakeLock != null )
         {
            this.mWakeLock.release();
            this.mWakeLock = null;
         }
      }
   }
```

Figure 1 Example Code of Wake Lock Misuse in The Open GPS Tracker App

The wake lock leak in Open GPS Tracker's GPSLoggerService.java was caused by improper wake lock management in the updateWakeLock() method. The issue arose because the method acquired a new partial wake lock every time GPS logging was active without checking whether an existing wake lock was already held. This led to multiple wake locks being acquired simultaneously, preventing the CPU from entering low-power states and causing excessive battery consumption. To address these issues, researchers have explored a range of techniques, including static analysis, Dynamic analysis, and machine learning, to detect and prevent energy leaks in mobile applications. Building on these techniques, Reverse engineering tools play a crucial role in identifying energy bugs in Android applications. Recent advancements include tools that monitor app states more accurately and machine learning models that predict potential energy drains, aiming to make diagnostics both proactive and precise. DroidLeaks is a comprehensive database addressing resource leaks in Android apps, which can degrade performance and cause crashes. Analyzing 124,215 code revisions from 34 open-source apps, it identified 292 fixed resource leak bugs across various classes. The study revealed common mismanagement patterns, such as the mistaken belief that only non-empty database cursors require closing, leading to leaks in apps/ These insights enhance static analysis tools for real-time detection and support the development of improved checkers, such as Android Lint plug-ins. By identifying fault patterns and solutions, DroidLeaks contributes to better resource management in Android development (Liu, Y et al.,2016; Arnatovich, Y. L. et al.,2018). This survey reviews the literature on energy inefficiency detection in Android applications, highlighting key methodologies and tools, such as dataflow analysis, state-taint tracking, and predictive machine learning models. By examining the strengths and limitations of these approaches, this survey seeks to provide a comprehensive overview of the current landscape, identify gaps in existing solutions, and suggest directions for future research toward more sustainable and energy-efficient mobile applications. Figure 1 shows Example Code of Wake Lock Misuse in The Open GPS Tracker App [1][2].

2. Literature Overview

Existing approaches for detecting performance issues in Android apps can be broadly classified into dynamic and static analysis methods. The application of machine learning and static analysis to detect energy inefficiencies in mobile applications, particularly in managing battery consumption within Android environments, has gained significant research attention over the years.

2.1 Early Foundations in Power Management

Pathak et al. identified "no-sleep" bugs in Android

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 03 Issue: 04 April 2025
Page No: 1625-1633
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2025.0231

apps caused by improper release of power resources, leading to battery drain. Their dataflow analysis tool effectively detected these bugs, highlighting the need for proper wake-lock management. Their early work laid the groundwork for understanding power consumption issues and highlighted the importance of wake-lock management in reducing battery drain. Liu et al. address energy inefficiency in Android applications caused by improper sensor and wake lock management [3-5]. To diagnose common energy issues, the authors developed GreenDroid, a tool for automated detection through app analysis. Building on this foundation, Liu et al. developed Elite, a static analysis tool that detects wake lock mismanagement in Android apps, helping prevent battery drain by identifying common misuse patterns (Pathak, A et al.,2012; Liu, Y et al.,2014; Liu, Y et al.,2016).

2.2 Static Analysis for Detecting Energy Leak

Jiang et al proposed a framework, named SAAD, that employs a combination of static analysis techniques to detect energy bugs by decompiling the APK files into Dalvik bytecode, followed by an inter-procedural analysis of the app. Wu et al. developed Relda2, a lightweight static analysis tool that constructs Function Call Graphs and Callback Graphs to capture inter-procedural interactions and Android callbacks, improving accuracy in detecting resource leaks by reducing false positives and negatives. Xu et al. advanced the field with Statedroid, employing state-taint analysis to track resource usage states and detect complex energy leaks involving multiple resources and system states in Android apps. Pereira extended EcoAndroid, an Android Studio plugin, to include an inter-procedural static analysis focusing on four Android resource types: Cursor, SQLite-Database, Wake-lock, and camera. Campelo et al developed E-APK (Energy-aware Android Patterns for Kadabra), a library of detectors designed to identify energy patterns in Java source code. This library leverages Kadabra's Abstract Syntax Tree (AST) model and a query-based search system to analyze both source and decompiled code, demonstrating that static analysis can be effectively applied even when the application's source code is unavailable (Jiang, H et al., 2015; Wu, T et al.,2017; Xu, Z et al., 2018; Pereria, R.B et al.,2022; Campelo, F.P et al.,2023).

2.3 Dynamic and Hybrid Analysis on Energy Leak Detection

Additional advancements in energy diagnostics are seen in tools like E-GreenDroid and Navydroid, which simulate Android app behaviors to monitor sensor and wake lock usage under different states. The researchers took a dynamic analysis approach by simulating how apps behave under different states, focusing on wake-lock and sensor usage. These tools detect inefficiencies through comprehensive execution models, enhancing diagnostic accuracy and scalability. EnergyPatch utilizes a hybrid analysis approach by combining static and dynamic analysis to detect and fix energy leaks in Android applications. It first employs static analysis to identify potential energy bugs, followed by dynamic analysis to validate these findings. Finally, it generates repair expressions to enhance app energy efficiency while addressing the challenges posed by extensive input domains. Abbasi et al. introduce the concept of application tail energy bugs (ATEBs) in smartphones, which occur when apps consume more power than expected or continue to use energy after being closed. It presents a Java-based tool designed to detect ATEBs by utilizing Android debug bridge commands to extract system-related information and evaluate its effectiveness through experiments with real Android apps (Wang, J et al.,2016; Liu, Y et al., 2017; Banerjee et al., 2018; Abbasi, A et al.,2018).

2.4 Machine Learning Approaches for Energy Leak Detection

Recently, Zhu et al. applied machine learning to energy bug detection, using a machine learning algorithm on system call data to predict high-energy-consumption code changes, thus enabling proactive identification of potential leaks. Khan et al. explored various machine learning algorithms to effectively detect energy leaks in Android applications, particularly focusing on wake-lock leaks. The proposed testing framework effectively detects energy issues in mobile applications by leveraging insights from an empirical study on energy inefficiencies. It employs well-designed input sequences and runtime contexts to trigger energy issues, particularly under conditions like poor

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 03 Issue: 04 April 2025
Page No: 1625-1633
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2025.0231

network performance. Machine learning techniques cluster workloads to differentiate essential processes from energy-draining ones. A systematic testing approach clears unrelated threads, selects candidate inputs, and monitors power consumption using a power monitor, logging detected issues into a database. The framework dynamically adjusts its testing based on detected issues, increasing the likelihood of uncovering further inefficiencies. With a time-budgeted testing process and practical implementations, it enhances energy issue detection, outperforming existing state-of-the-art methods in large-scale evaluations (Zhu, C et al.,2019; Khan, M.U et al.,2021; Li, X et al.,2022).

2.5 Estimation of Power Consumption in Mobile Apps

Le et al. introduces a novel approach to modeling and estimating power consumption in Android apps by defining power states for key hardware components like GPS, Wi-Fi, CPU, and display. Using a power consumption automaton (PCA), the framework accurately estimates energy use and optimizes power states through algorithmic refinements. By merging power states into a unified model, this approach helps developers visualize and enhance app efficiency, improving user experience and battery life (Le, H. A et al.,2019).

3. Methodologies and Approaches

The detection of energy inefficiencies, particularly wake-lock leaks, in Android apps has been addressed using various methodologies and tools over the years. These approaches primarily focus on static analysis, dynamic analysis, and machine learning techniques to detect patterns of misuse and predict energy bugs. Below is a detailed explanation of the methodologies adopted by key studies in the field of mobile energy management [7-10].

3.1 Dataflow-based Static Analysis for Wake-lock Leak Detection

One of the earliest methodologies used for detecting wake-lock leaks was developed by Pathak et al. developed a dataflow-based static analysis tool that detects bugs by identifying unmatched wake lock acquire and release calls, specifically targeting paths where components are left unnecessarily active on applications by collecting data on battery-draining issues from bug reports, mobile forums, and code repositories (Pathak, A et al.,2012).

3.2 Elite: Static Analysis Tool for Wake-lock Leak Mismanagement

Liu et al. analyzed common misuse patterns of wake locks in Android apps by examining API calls for acquiring and releasing wake locks across app codebases and they created Elite, a static analysis tool that uses dataflow analysis to automatically identify wake lock mismanagement without relying on predefined assumptions about usage contexts. Elite analyzed common misuse patterns of wake locks by examining the API calls for acquiring and releasing wake locks across the app's codebase. Unlike earlier approaches, Elite did not rely on predefined assumptions about the context of wake-lock usage (Liu, Y et al.,2016). [6]

3.3 Relda2: A Lightweight and Precise Static Analysis Tool for Resource Leak Detection

Relda2 adopts a two-pronged analysis technique: flow-insensitive and flow-sensitive analysis. Flow-insensitive analysis quickly scans bytecode sequentially to identify resource requests and release operations, sacrificing precision for speed. In contrast, flow-sensitive analysis constructs Control Flow Graphs (CFGs) and Value Flow Graphs (VFGs) to preserve control flow information, enabling more accurate detection of resource leaks with fewer false negatives. The tool also employs lightweight inter-procedural analysis and optimizes performance through multi-threading, ensuring scalability (Wu, T et al.,2016).

3.4 Dynamic Simulation Models for App Behavior

Wang et al. included simulating app behavior to monitor wake lock and sensor states. They developed E-GreenDroid, which uses the Android Execution Model (AEM) to simulate app state transitions and identify patterns of wake lock mismanagement and sensor data underutilization. E-GreenDroid's dynamic analysis provided a more detailed view of how an app interacts with the system in real-world conditions, such as how the app manages sensors or wake locks during different execution states (Wang, J et al.,2016).

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 03 Issue: 04 April 2025
Page No: 1625-1633
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2025.0231

3.5 NavyDroid: Extended Execution Model for Complex Energy Bugs

Building on the E-GreenDroid methodology, NavyDroid extended the E-GreenDroid approach by refining its execution model to capture more accurate app state transitions. Its methodology relied on deterministic finite automata (DFA) to simulate Android app state changes precisely which applies a precise execution model to simulate app state behavior and identify complex wake lock misuse patterns (Liu, Y et al.,2017).

3.6 EnergyPatch: Hybrid Approach for Repairing Resource Leaks

The EnergyPatch framework employs a combination of static and dynamic analysis techniques to effectively detect, validate, and repair energy bugs in Android apps. The methodology begins with a lightweight static analysis that quickly narrows down potential program paths where energy bugs may occur [11]. This initial step is crucial as it allows for efficient identification of problematic areas without exhaustive exploration, which is often impractical due to the vast input domains of mobile apps. Following the static analysis, the framework utilizes dynamic analysis techniques to explore the identified potentially buggy program paths. This exploration helps in validating the reported energy bugs and generating relevant test cases that can be used for analysis. Finally, EnergyPatch generates repair expressions aimed at fixing the validated energy bugs (Banerjee, A et al.,2018).

3.7 State-Taint Analysis for Resource Management

State-taint analysis was introduced to track resource management in Android applications. Their methodology involved tracking resource allocation and release operations in type state protocols to detect resource mismanagement patterns. They implemented State droid, a tool that applies taint tracking to analyze resource state transitions in Android code, enabling the detection of complex resource bugs. By employing state-taint analysis, Statedroid can identify more complex resource management patterns and detect energy leaks that involve multiple resources and system states (Xu, Z et al.,2018).

3.8 Machine Learning for Predicting Energy Bugs

Zhu et al. used machine learning to predict energy bugs by analyzing system call data from various app revisions. Their methodology involved selecting relevant features through recursive feature elimination and tuning model parameters with cross-validation. They applied Lasso regression to build a predictive model that identifies code changes leading to increased energy consumption. This approach emphasized using historical revision data to proactively identify energy-inefficient code, marking a shift towards predictive diagnostics in mobile energy management. In particular, Khan et al. proposed a machine learning-based detection model that leverages function call graphs extracted from APK files. By employing a range of machine learning algorithms, including Support Vector Machine (SVM) and Stochastic Gradient Boosting (SGB), the model achieved high accuracy rate in detecting wake lock leak respectively. This demonstrates the potential of machine learning in accurately detecting wake lock leaks by recognizing complex patterns in large-scale Android applications (Zhu, C et al.,2019; Khan, M.U et al.,2021).

4. Findings and Trends

Over the years, the detection of energy inefficiencies in mobile applications, particularly in Android apps, has evolved significantly. Early research laid the foundation for identifying common power issues, while subsequent studies introduced more advanced techniques, expanding the scope of detection and improving diagnostic accuracy.

4.1 Shift Toward Predictive Energy Diagnostics with Machine Learning

Recent developments that involve integrating machine learning into energy diagnostics. By applying Lasso regression to system call data, this approach shifts from reactive to proactive identification of potential energy leaks, enabling developers to detect high-energy-consumption code changes early in the development process. This trend marks a growing interest in predictive energy diagnostics to address inefficiencies before they impact users. By analyzing system call data and using recursive feature elimination, their approach could

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 03 Issue: 04 April 2025
Page No: 1625-1633
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2025.0231

predict high-energy-consumption code changes, enabling developers to address energy bugs before they impact users [12-15]. This predictive shift marks a significant trend in the field, as researchers aim to use machine learning models to not only diagnose energy inefficiencies but also forecast potential problems, allowing for earlier intervention (Zhu, C et al., 2019).

4.2 Automated Detection and Repair of Android Resource Leaks and Code Smells

PlumbDroid uses static analysis and resource-flow graphs to detect leaks, fixing 26 issues in 17 apps in under 80 seconds per leak, though it struggles with aliasing. Similarly, Fatima et al. developed a framework that identifies and corrects lifecycle misuses, wake locks, and context leaks through rule-based detection and automated fixes. Both works emphasize the value of automated repair in Android's event-driven architecture (Bhatt et al., 2020; Fatima, I et al., 2020).

4.3 Energy Consumption Analysis in Android Applications

Ahmed et al analyzed energy usage trends in Kotlin-based Android apps, revealing increased consumption linked to OS upgrades, UI issues, and inefficient use of features like coroutines and Data Binding. Groza et al introduced a framework to detect and fix energy smells such as HashMap usage and unnecessary getters/setters resulting in up to 30% battery savings. Together, these studies highlight the importance of analyzing and addressing energy inefficiencies for more sustainable mobile applications (Ahmed, H et al., 2023; Groza, C et al.,2024).

5. Challenges and Gaps

Despite significant advancements in detecting energy inefficiencies in Android apps, several challenges and gaps remain. This section discusses the limitations faced by current tools and methodologies, particularly when addressing the complex, context-dependent, and concurrent nature of mobile applications.

5.1 Static Analysis Tools Are Subject to Energy Inefficiency Detection Limitations

Static analysis tools such as Elite and E-GreenDroid are useful for detecting basic wake lock misuses and "no-sleep" bugs but often fall short when analyzing complex, context-sensitive energy inefficiencies. Due to the event-driven and dynamic nature of mobile applications, these tools struggle to capture runtime dependencies influenced by user interactions or app states, resulting in higher false positive rates. Table 1 shows Performance Metrics of Energy Leak Detection Tools. This limitation is reflected in the performance metrics of various energy leak detection tools, as shown in Table 1, and tools like Relda2 and Ecodroid are subjected to relatively high false positive rates, highlighting the challenges faced by static analysis in accurately identifying energy-related issues. (Pathak, A et al.,2012; Wang, J et al., 2016, Lia, L et al 2017).

Table 1 Performance Metrics of Energy Leak Detection Tools

RESEARCH	ACCURACY/PRECISION (%)	FALSE POSTIVE (%)
SAAD	86.67	13.33
StateDroid	83.5	19
Zhu et al	93.4	-
Khan et al	98	7.2

5.2 Challenges in Concurrent Operations and Multi-Threading

The asynchronous and multi-threaded nature of many Android applications presents a challenge for static analysis tools, which are often limited in their ability to model concurrent operations accurately. This limitation can lead to missed or incorrect diagnoses of energy inefficiencies when multiple threads or background services independently manage resources like wake-locks [16].

5.3 Scalability Issues in Stateful Analysis Tools

Tools such as NavyDroid, which simulate comprehensive app states, may encounter scalability issues, especially when applied to large or complex applications. The computational overhead associated with stateful analysis, particularly in the presence of nested resource states, can impede practical adoption and limit these tools' real-time diagnostic capabilities (Liu, Y et al., 2017).

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 03 Issue: 04 April 2025
Page No: 1625-1633
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2025.0231

5.4 Data Limitations for Machine Learning Models

Machine learning methods, such as the Lasso regression and support vector machine require extensive, high-quality labeled data for effective training [17-21]. Obtaining sufficient high-quality training data from diverse applications is resource-intensive, and its data limitation affects the predictive accuracy (Zhu, C et al., 2019; Khan, M. U et al., 2021).

Conclusion

This survey provides a comprehensive overview of the advancements in energy leak detection techniques for Android applications, with a particular emphasis on identifying wake-lock leaks. The literature highlights how traditional static analysis methods have significantly contributed to our understanding of resource management issues. However, as Android applications grow more complex, these methods face limitations in detecting dynamic and context-dependent resource leaks effectively. Consequently, there is a notable shift in recent research toward leveraging machine learning models that analyze app performance data to detect energy inefficiencies. Given these trends, this survey supports the adoption of machine learning algorithms as an advanced approach to energy leak detection. Machine learning enables the analysis of diverse and complex app usage patterns, making it possible to detect wake-lock leaks and other resource-related issues with greater accuracy and adaptability. By integrating app performance data into these models, this approach promises a more scalable and robust solution, aligning with the evolving needs of mobile application development. This study thus lays the groundwork for the practical implementation of machine learning in energy leak detection, marking an essential step forward in optimizing app efficiency and user experience [22-25].

References

[1]. Pathak, A., Jindal, A., Hu, Y. C., & Midkiff, S. P. (2012). What is keeping my phone awake? Characterizing and detecting no-sleep energy bugs in smartphone apps. In Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys '12), Lake District, UK. ACM. https://dl.acm.org/doi/10.1145/2307636.2307661

[2]. Liu, Y., Xu, C., Cheung, S. C., & Terragni, V. (2016). Understanding and detecting wake lock misuses for Android applications. In Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering (SLE '16). ACM. https://dl.acm.org/doi/10.1145/2950290.2950297

[3]. Wang, J., Liu, Y., Xu, C., Ma, X., & Lu, J. (2016). E-GreenDroid: Effective energy inefficiency analysis for Android applications. In Proceedings of Internetware '16, Beijing, China. ACM. https://dl.acm.org/doi/10.1145/2993717.2993720

[4]. Liu, Y., Wang, J., Xu, C., & Ma, X. (2017). NavyDroid: Detecting energy inefficiency problems for smartphone applications. In Proceedings of Internetware 2017, Shanghai, China. ACM. https://dl.acm.org/doi/10.1145/3131704.3131705

[5]. Xu, Z., Wen, C., & Qin, S. (2018). State-taint analysis for detecting resource bugs. Science of Computer Programming, 162, 93-109. Elsevier. https://doi.org/10.1016/j.scico.2017.06.010

[6]. Zhu, C., Zhu, Z., Xie, Y., Jiang, W., & Zhang, G. (2019). Evaluation of machine learning approaches for Android energy bug detection with revision commits. IEEE Access, 7, 85241–85252. https://doi.org/10.1109/ACCESS.2019.2924953

[7]. Khan, M. U., Lee, S. U., Abbas, S., Abbas, A., & Bashir, A. K. (2021). Detecting wake lock leaks in Android apps using machine learning. IEEE Access, 9. https://doi.org/10.1109/ACCESS.2021.3110244

[8]. Banerjee, A., & Roychoudhury, A. (2015). EnergyPatch: Repairing resource leaks to

improve energy efficiency of Android apps. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (FSE), 37-49. https://doi.org/10.1145/2786805.2786827

[9]. Palomba, F., Di Nucci, D., Panichella, A., Zaidman, A., & De Lucia, A. (2019). On the impact of code smells on the energy consumption of mobile applications. Information & Software Technology, 105(105), 43–55. https://doi.org/10.1016/J.INFSOF.2018.08.004.

[10]. Wu, T., Liu, J., Xu, Z., Guo, C., Zhang, Y., Yan, J., & Zhang, J. (2016). Light-weight, inter-procedural, and callback-aware resource leak detection for Android apps. IEEE Transactions on Software Engineering. https://doi.org/10.1109/TSE.2016.2547385

[11]. Arnatovich, Y. L., Wang, L., Ngo, N. M., & Soh, C. (2018). A comparison of Android reverse engineering tools via program behaviors validation based on intermediate languages transformation. IEEE Access, 6. https://doi.org/10.1109/ACCESS.2018.2808340

[12]. Abbasi, A. M., Al-Tekreeti, M., Naik, K., Nayak, A., Srivastava, P., & Zaman, M. (2018). Characterization and detection of tail energy bugs in smartphones. IEEE Access, 6. https://doi.org/10.1109/ACCESS.2018.2877395

[13]. Jiang, H., Yang, H., Qin, S., Su, Z., Zhang, J., & Yan, J. (2017). Detecting energy bugs in Android apps using static analysis. In Proceedings of the IEEE International Conference. https://doi.org/10.1007/978-3-319-68690-5_12

[14]. Pereira, R. B., Ferreira, J. F., Mendes, A., & Abreu, R. (2022). Extending EcoAndroid with Automated Detection of Resource Leaks. International Conference on Mobile Software Engineering and Systems, 17–27. https://doi.org/10.1145/3524613.3527815

[15]. Campelo, F. P., Sousa, M. C. B. de O., & Nascimento, C. L. (2023). E-APK: Energy pattern detection in decompiled android applications. Journal of Computer Languages, 76, 101220. https://doi.org/10.1016/j.cola.2023.101220

[16]. Khan, M. U., Abbas, S., Lee, S. U.-J., & Abbas, A. (2020). Energy-leaks in Android application development: Perspective and challenges. Journal of Theoretical and Applied Information Technology, 98(22), 2005–ongoing.

[17]. Liu, Y., Xu, C., Cheung, S.-C., & Lu, J. (2014). GreenDroid: Automated Diagnosis of Energy Inefficiency for Smartphone Applications. IEEE Transactions on Software Engineering, 40(9), 911–940. https://doi.org/10.1109/TSE.2014.2323982

[18]. H. Ahmed et al., "Evolution of Kotlin Apps in terms of Energy Consumption: An Exploratory Study," 2023 International Conference on ICT for Sustainability (ICT4S), Rennes, France, 2023, pp. 46-56, doi: 10.1109/ICT4S58814.2023.00014.

[19]. Groza, C., Dumitru-Cristian, A., Marcu, M., & Bogdan, R. (2024). A Developer-Oriented Framework for assessing power consumption in mobile Applications: Android Energy Smells Case Study. Sensors, 24(19), 6469. https://doi.org/10.3390/s24196469

[20]. Fatima, I., Anwar, H., Pfahl, D., Qamar, U., College of Electrical and Mechanical Engineering, National University of Sciences and Technology, & Institute of Computer Science, University of Tartu. (2020). Detection and correction of Android-specific code smells and energy bugs: An Android Lint extension. QuASoQ 2020: 8th International Workshop on Quantitative Approaches to Software Quality, 71

[21]. Sahin, C. (2024). Do popular apps have issues regarding energy efficiency? PeerJ, 10, e1891. https://doi.org/10.7717/peerj-cs.1891

[22]. Li, X., Chen, J., Liu, Y., Wu, K., & Gallagher, J. J. (2022). Combatting Energy Issues for Mobile Applications. ACM Transactions on Software Engineering and Methodology, 32(1), 1–44.

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 03 Issue: 04 April 2025
Page No: 1625-1633
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2025.0231

https://doi.org/10.1145/3527851

[23]. Bhatt, B. N., & Furia, C. A. (2020). Automated Repair of Resource Leaks in Android Applications. arXiv: Software Engineering. https://doi.org/10.1016/j.jss.2022.111417

[24]. Liu, Y., Wei, L., Xu, C., & Cheung, S.-C. (2016). DroidLeaks: Benchmarking Resource Leak Bugs for Android Applications. arXiv: Software Engineering. https://dblp.uni-trier.de/db/journals/corr/corr1611.html#LiuWXC16

[25]. https://github.com/rcgroot/opengpstracker/blob/8ac7905a5ac78520c63adb864eb0765eca08cc56/application/src/nl/sogeti/android/gpstracker/logger/GPSLoggerService.java