International Journal of Computer Science and Information Technology Research (IJCSITR)
2024, Vol. 5, No. 4, October- December, pp. 1-16
Journal ID: 9471-1297
website: www.ijcsitr.com

Power Restrictions for Android OS: Managing Energy Efficiency and System Performance

Darshan Mohan Bidkar[1], Vivekananda Jayaram[2], Manjunatha Sughaturu Krishnappa[3], Amey Ram Banarse[4], Gaurav Mehta[5], Koushik Kumar Ganeeb[6], Shenson Joseph[7], Prema kumar Veerapaneni[8],

[1]ORCID: 0009-0001-0057-0527, [2]ORCID: 0009-0004-9389-9074,[3]ORCID: 0009-0009-5260-0503, [4]ORCID: 0009-0001-1515-3240, [5]ORCID: 0009-0005-0911-3081, [6]ORCID: 0009-0002-3751-3195, [7]ORCID: 0009-0001-5191-5556, [8]ORCID: 0009-0003-5421-8515

Abstract

Power management is critical for enhancing the performance and longevity of mobile devices running the Android operating system. This paper examines the evolution of power restrictions in Android OS, focusing on recent advancements in energy-efficient application management. We analyze key mechanisms such as Doze mode, App Standby, and background execution limits, and their impact on extending battery life while maintaining a balance between user experience and system performance. The paper also address the challenges developers encounter in optimizing applications within these constraints, offering insights into best practices and future pathways for enhancing energy efficiency in Android-powered devices.

Keywords:

Android OS power management, Efficient data fetching and syncing, AI-driven power optimization, Context-aware energy management, Energy efficiency in mobile devices

How to Cite: Bidkar, D.M., Jayaram, V., Krishnappa, M.S., Banarse, A.R., Mehta, G., Ganeeb, K.K., Joseph, S., Veerapaneni, P.K. (2024). Power Restrictions for Android OS: Managing Energy Efficiency and System Performance. *International Journal of Computer Science and Information Technology Research*, 5(4), 1–16. DOI: https://doi.org/10.5281/zenodo.14028551

Article Link: https://ijcsitr.com/index.php/home/article/view/IJCSITR_2024_05_04_01/IJCSITR_2024_05_04_01

1. Introduction

The widespread adoption of smartphones has led to an increasing demand for devices that are both powerful and energy-efficient. Battery life is a key consideration for users, as longer battery life enhances the usability and convenience of mobile devices. The Android operating system, which is one of the most widely used mobile platforms, has implemented various power management strategies to address these user needs.

Early versions of Android featured basic power-saving measures, such as limiting background processes and adjusting screen brightness, but these methods were insufficient as applications grew more complex and power-hungry. In response, Google introduced more sophisticated power management features in subsequent Android versions to enhance battery performance while minimizing disruptions to user experience [1, 13,14].

This paper aims to provide a comprehensive overview of the evolution of power-saving mechanisms in Android, including Doze mode, App Standby, and background execution limits [6]. We examine the effectiveness of these features in reducing energy consumption and explore their impact on both application performance and battery longevity. Additionally, the paper presents guidelines for developers to optimize their applications for energy efficiency while maintaining the desired level of performance [13, 14].

2. Literature Review

2.1. Overview of Power Management in Android OS

Power management in Android OS has evolved significantly over the years, with each new version introducing features to optimize energy consumption. Google's documentation highlights mechanisms such as Doze mode, App Standby, and background execution limits introduced in recent Android versions [1]. However, a thorough evaluation of their impact on application behavior and user experience remains a subject of ongoing research. Studies have shown the benefits and challenges of these power management mechanisms [2, 3]. Evolution of Android power management is shown in Fig.1.

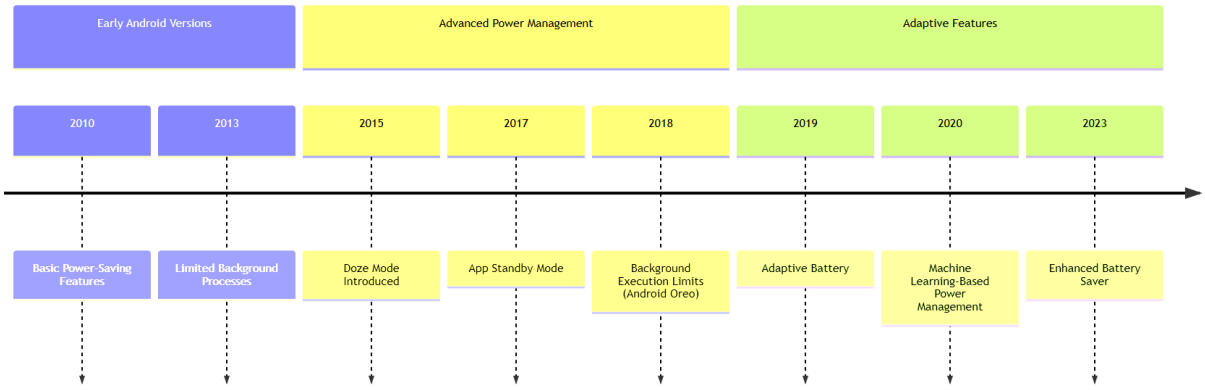

Figure 1. Android power management evolution

2.2. Comparative Studies on Power Optimization Techniques

Optimizing Android power consumption involves analyzing various strategies employed by Android OS [4]. User behavior patterns, such as device usage and idle times, play a role in determining power management techniques [5]. Marimuthu et al. highlighted that while Doze mode and App Standby effectively extend battery life during idle states, they may delay background tasks, affecting real-time applications. This paper proposes a balanced approach to maintain app functionality while still benefiting from power-saving features, focusing on dynamic app behavior adaptation [6][7].

Table-1. Literature survey summary

Author(s)	**Year**	**Focus Area**	**Key Findings**
Marimuthu et al.	2021	Doze Mode and App Standby impacts on real-time applications	Doze Mode and App Standby extend battery life but may delay background tasks.
Hurbungs et al.	2016	Trade-offs between resource efficiency and performance in Android 8.0	Background limits improve battery life but hinder adaptation for real-time demands.
Gupta et al.	2024	Battery optimization in Android applications	Proposed solutions to reduce battery consumption for various app categories.
Souza et al.	2024	Energy consumption optimization techniques	Recommendations for reducing energy consumption in mobile applications.
Ruiz Nepomuceno et al.	2024	Systematic review on adaptive mobile learning systems	Evaluated various architectures for adaptive learning with power optimization.
Awad et al.	2024	Battery drainage minimization techniques	Developed a recommendation system to minimize battery drainage in smartphones.

2.3. Android's Background Execution Limits

Recent Android versions have introduced background execution limits, crucial for managing power consumption [8, 9]. Hurbungs et al. research, focusing on Android 8.0 (Oreo), discusses

trade-offs between resource efficiency and application performance [11, 16-18]. Limiting background services improves battery life, but applications struggle to adapt under real-time demands. This paper builds on Hurbungs' findings by offering practical solutions for developers to use WorkManager and job schedulers to mitigate background execution limits [10].

2.4. Gap in Current Literature

Previous studies by Marimuthu and Hurbungs provide insights into Android's power management techniques, but a gap remains in practical guidelines for optimizing apps under new constraints [13]. This paper addresses this gap by proposing a framework for developers to adapt their applications to the evolving power management landscape, balancing energy efficiency and app performance without compromising essential functionality. Summary of the literature survey is as shown in Tab-1.

3. Proposed Approach

The proposed work aims to provide a more developer-centric approach compared to previous studies, offering clear strategies for optimizing applications under Android's power restrictions. By synthesizing insights from the literature, this paper presents an approach leveraging existing power-saving features while minimizing their impact on application functionality. Specifically, we discuss how developers can use Android APIs like WorkManager, JobScheduler, and the Adaptive Battery framework to optimize energy efficiency [4].

4. Power Management in Android OS

4.1 Early Power Management Techniques

Android OS initially used basic power-saving techniques like limiting background processes and reducing screen brightness [14]. However, as applications grew complex, more sophisticated approaches became necessary [16][17]. Optimization strategies for developers are as shown in Fig. 2.

4.2. Doze Mode and App Standby

Introduced in Android 6.0, Doze mode and App Standby restrict network and processing resources for inactive apps, significantly extending battery life [12, 15]. Research by Marimuthu et al. shows Doze mode's effectiveness in battery optimization, though its benefits vary based on user behavior and app category [6].

4.2.1. Optimization Strategy for Doze Mode:

a) *Use High-Priority FCM Messages:* Developers can use Firebase Cloud Messaging (FCM) to send high-priority messages that can wake the device from Doze Mode. This is useful for apps that need to notify users in real time, like messaging or emergency apps. However, developers should be cautious not to misuse high-priority messages as excessive use can degrade battery performance and lead to app uninstallation.

b) *JobScheduler API for Deferred Tasks:* Apps should defer non-urgent tasks by

scheduling jobs using the JobScheduler API. Jobs scheduled with *setRequiresDeviceIdle*() can run during maintenance windows in Doze Mode, allowing the system to bundle tasks and optimize battery use.

c) *Optimize AlarmManager Usage:* AlarmManager should only be used when absolutely necessary, as alarms are deferred during Doze Mode unless they are high-priority alarms. If an alarm is critical, it can be scheduled with *setExactAndAllowWhileIdle*(), but such usage should be minimized to avoid draining the battery.

d) *Network Batching and Caching:* Since Doze Mode restricts network access, developers can optimize their apps by batching network operations and caching data when the device is active. Apps should synchronize data during the brief maintenance windows that occur periodically in Doze Mode.

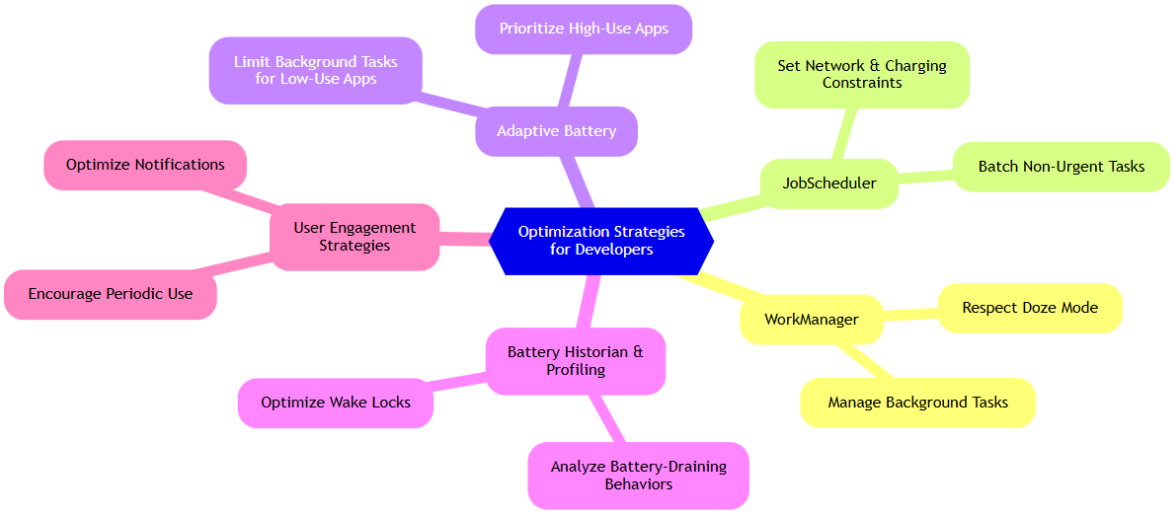

Figure 2. Optimization Strategies for Developers

4.2.2. Optimization Strategy for App Standby:

a) *Use WorkManager for Background Tasks:* WorkManager is a robust solution for managing background tasks that ensures tasks are completed while respecting system restrictions. WorkManager can manage constraints such as network availability and charging status and can queue tasks to be executed once the device exits App Standby or Doze Mode.

b) *Respect App Standby Buckets:* Starting with Android 9 (Pie), apps are classified into standby buckets based on usage frequency (Active, Working Set, Frequent, and Rare). Apps should avoid unnecessary background processing to stay in higher buckets (Active or Working Set) and improve performance while minimizing battery drain. Developers can monitor app bucket status through the *UsageStatsManager* API and adapt their app behavior accordingly.

c) *Incorporate User Engagement Strategies:* To prevent apps from being placed in standby, developers can implement features that encourage periodic user engagement. For

example, apps can send notifications about new content, but these notifications should be meaningful and non-intrusive to avoid annoying the user and leading to uninstallation.

d) *Handle Background Services with Care:* Background services are a common source of power drain, and Android limits their use in App Standby. Developers should replace persistent background services with scheduled jobs or use foreground services when real-time tasks are necessary. Foreground services display a persistent notification, which informs users about the app's background activity and prevents excessive power consumption.

5. Background Execution Limits

Android introduced background execution limits in Android 8.0 (Oreo) to manage energy consumption of applications, especially those performing background tasks without user interaction. These limits restrict apps from running background services when not in active use, requiring developers to rethink how they manage background activities without impacting performance [11-13]. This section proposes a suite of developer tools and strategies to balance performance with compliance to these limits.

5.1. WorkManager

WorkManager is Android's recommended solution for handling background tasks that require guaranteed execution, regardless of app state or device reboot. It simplifies managing background jobs by respecting system restrictions, such as Doze Mode and background execution limits, while allowing tasks to be deferred or batched based on the system's power state.

5.1.1. Advantages of WorkManager:

a) *Guaranteed Task Completion:* Ensures tasks complete even if the app is terminated or the device restarts, ideal for tasks like periodic data syncs and logging.

b) *Chaining and Constraining Work:* Enables chaining multiple tasks and setting constraints, such as network or charging requirements, optimizing battery usage by executing tasks in optimal states.

c) *Flexibility in Execution:* Allows developers to defer tasks until the device exits battery-saving modes, suitable for tasks that can wait for network availability.

5.1.2. Optimization Strategy with WorkManager:

- Use *OneTimeWorkRequest* for tasks needing a single execution.
- Use *PeriodicWorkRequest* for periodic tasks, such as refreshing server data every 24 hours.
- Chain related tasks using *WorkContinuation* for sequential or parallel execution, adhering to device power constraints.

5.2. JobScheduler

JobScheduler, introduced in Android 5.0 (Lollipop), enables developers to schedule jobs based on conditions like network availability, charging state, and idle status. JobScheduler defers background work, optimizing battery life by grouping jobs from different apps to run in batches, reducing system wake-ups.

5.2.1. Advantages of JobScheduler:

- *Batch Execution:* Schedules jobs to run together, reducing CPU wake-ups and optimizing battery consumption.
- *Device-Specific Constraints:* Allows conditions like *"run only on Wi-Fi"* or *"run only when idle"*, ensuring background tasks operate in power-efficient scenarios.
- *Automatic Re-execution:* Jobs failing due to unmet conditions are retried automatically, ensuring essential background tasks complete eventually without affecting user experience.

5.2.2. Optimization Strategy with JobScheduler:

- Schedule non-urgent tasks like syncing logs or backups using *JobInfo.Builder* with constraints like *setRequiresCharging*() and *setRequiresDeviceIdle*().
- Use *JobInfo.setMinimumLatency*() and *setOverrideDeadline*() to defer tasks or ensure timely execution, balancing performance and energy use.

5.3. Foreground Services

For apps needing continuous background execution, such as media players or GPS navigation, Android provides Foreground Services, exempt from background execution limits but requiring a persistent notification to inform users of their operation [16, 17].

5.3.1. Advantages of Foreground Services:

- *Real-Time Execution:* Ideal for tasks requiring continuous real-time operation, such as playing music or tracking location.
- *Persistent Notifications:* Provides transparency, allowing users to stop the service if no longer needed.

5.3.2. Optimization Strategy with Foreground Services:

a) Use foreground services only for tasks needing continuous background activity and immediate execution.

b) Ensure the persistent notification provides clear, actionable information, like an option to stop the service.

c) Minimize foreground service use by transitioning to WorkManager or JobScheduler when real-time processing is unnecessary.

5.4. Broadcast Receivers and Pending Intents

Broadcast Receivers and Pending Intents enable developers to schedule tasks based on specific system events, like device reboot or network availability. These tools are essential for

apps needing to listen to system-wide events without continuous background activity.

5.4.1. Advantages of BroadcastReceivers and PendingIntents:

a) *Event-Driven Execution:* Efficiently performs background tasks only when necessary by responding to system broadcasts (e.g., ACTION_BOOT_COMPLETED), reducing battery drain.

b) *Minimal Resource Consumption:* Remain passive until activated by the specific event they monitor.

5.4.2. Optimization Strategy with BroadcastReceivers and PendingIntents:

a) Use PendingIntent to schedule alarms, notifications, or service starts without running continuous background processes.

b) Limit the use of implicit broadcasts (restricted from Android 8.0 onward) and register for explicit broadcasts or use JobScheduler for essential background work triggered by events.

5.5. Battery Historian and Profiling Tools

Battery Historian is a powerful tool for identifying battery-draining behavior in apps. By analyzing wake locks, network usage, and background activity, developers can optimize app performance to comply with Android's background execution limits while maximizing battery life.

5.5.1. Advantages of Battery Historian:

a) *Detailed Performance Metrics:* Provides extensive logs on app behavior during background execution, like wake lock usage, job scheduling efficiency, and Doze Mode performance.

b) *Troubleshooting Power Drains:* Pinpoints specific components causing battery drain, allowing targeted optimization or refactoring.

5.5.2. Optimization Strategy with Battery Historian:

- Regularly profile the app during testing to identify and resolve battery-draining behaviors.
- Minimize wake locks and background network usage to comply with background execution limits and improve battery efficiency.

5.6. Adaptive Battery API

The Adaptive Battery API, introduced in Android 9 (Pie), uses machine learning to prioritize battery usage based on user interaction patterns. Apps less frequently accessed face restricted background resource usage, while frequently used apps receive more leniency.

5.6.1. Advantages of Adaptive Battery:

a) *Intelligent Battery Optimization:* Optimizes background tasks according to app usage frequency, reducing battery drain for infrequently used apps without intervention.

b) *Auto-Management of Background Tasks:* Automatically restricts less frequently used apps, enabling developers to focus on optimizing high-priority tasks.

5.6.2. Optimization Strategy with Adaptive Battery:

- Respect background execution limits enforced by Adaptive Battery to avoid excessive resource consumption for infrequent app usage.
- Encourage users to whitelist critical real-time services, like fitness tracking or emergency alerts, ensuring reliable functionality in low-power conditions.

By leveraging these tools and strategies, developers can align with Android's background execution limits, optimizing both app performance and battery efficiency while ensuring a positive user experience under power-saving constraints..

6. Challenges & Best Practices for Developers

Adapting applications to comply with Android's power restrictions presents significant challenges for developers. Background services, essential for app functionality, may not operate as expected under these limits. This section outlines strategies to address these challenges, particularly through the use of job schedulers and WorkManager APIs. Schematic layout for Challenges and best practices for developers are as shown in Fig.3.

6.1. Power-Sensitive User Settings

6.1.1. Impact on User Experience:

User preferences for power usage vary widely—some prioritize real-time updates and performance, while others aim to extend battery life as much as possible [17]. A universal approach to power settings may leave users unsatisfied, especially those who prefer control over app power usage.

6.1.2. Solution: Offer Power-Sensitive Settings for Users.

Provide customizable settings for background activity, sync frequency, and notifications, enabling users to balance performance and power consumption as per their needs.

6.1.3. Best Practices:

- Allow users to toggle background sync, update frequency, and notifications, with preset modes such as *"High Performance"*, *"Balanced",* and *"Battery Saver"*.
- Enable users to disable background tasks when the battery is low, letting them optimize performance based on immediate needs.
- Clearly communicate the impact of each setting on battery usage and performance, so users understand the trade-offs.

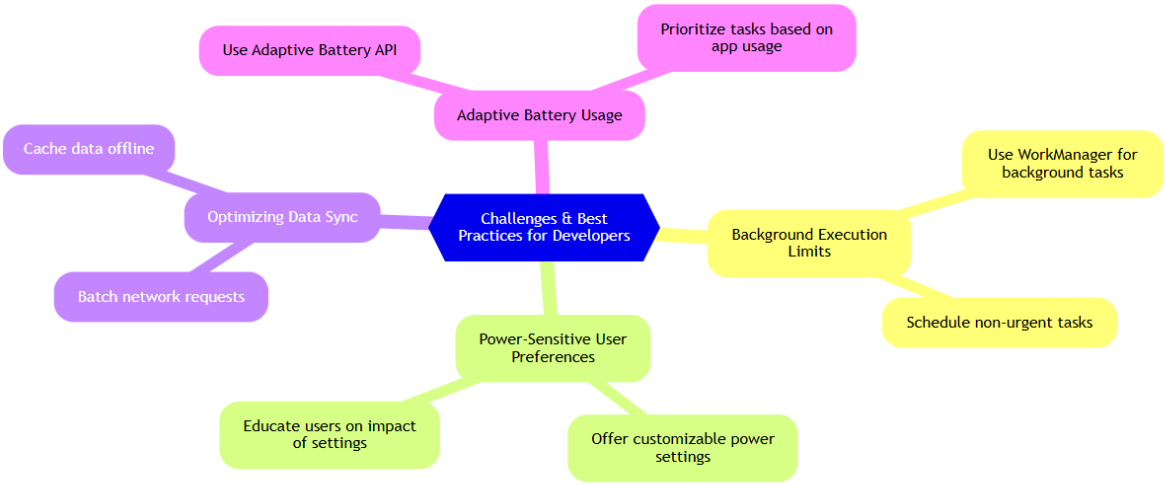

Figure 3. Challenges and best practices for developers

6.2. Efficient Data Fetching and Syncing

6.2.1. Impact on User Experience:

Frequent data fetching, like real-time updates or syncing, can drain battery life quickly [18]. Reducing update frequency may lead to outdated content, frustrating users who expect timely information (e.g., news feeds, stock updates).

6.2.2. Solution: Batch Network Requests and Implement Data Caching

Batching network requests reduces battery drain, and local caching enables content display even offline or when updates are delayed.

6.2.3. Best Practices:

- Batch network requests to minimize background activity, such as fetching data only once an hour or when the app is open.
- Cache critical data locally to ensure users can access recent content even if real-time updates are delayed.
- Use *WorkManager* or *JobScheduler* for periodic data syncing during optimal conditions, like Wi-Fi connection or charging, to enhance energy efficiency.

6.3. Adaptive Battery and App Standby Buckets

6.3.1. Impact on User Experience:

With Android 9 (Pie), Adaptive Battery uses machine learning to limit background activity for infrequently used apps [19]. Apps are categorized into standby buckets (Active, Working Set, Frequent, Rare) based on usage. Apps in lower buckets face more background restrictions, which can impact performance when users return to them.

6.3.2. Solution: Encourage User Engagement to Maintain Higher Standby Buckets

By providing relevant notifications and features that align with user habits, developers can encourage engagement that keeps the app in a higher standby bucket, allowing more background resources.

6.3.3. Best Practices:

- Drive user engagement through reminders, content updates, or features prompting users to open the app, helping maintain higher-priority buckets.
- Ensure notifications are valuable and actionable but avoid excessive alerts that may lead to uninstalls.
- Use Google Play's Pre-launch report to test the app's behavior on different devices under Adaptive Battery settings, ensuring a consistent user experience.

6.4. Testing and Profiling Tools

6.4.1. Impact on User Experience:

Balancing performance with power efficiency without impacting user experience is challenging. Testing app behavior under power restrictions, such as Doze Mode or App Standby, is crucial for optimization.

6.4.2. Solution: Use Android's Testing and Profiling Tools

Android offers tools like Battery Historian, ADB Shell commands, and Profiler to identify and optimize performance bottlenecks related to power consumption. These tools help developers pinpoint excessive background activity, wake locks, and delayed tasks for battery efficiency.

6.4.3. Best Practices:

- Regularly use Battery Historian to analyze battery usage and identify excessive background activities, ensuring optimal app performance.
- Simulate Doze Mode and App Standby using commands like *adb shell dumpsys battery set level* and *adb shell cmd appops set* <PACKAGE> RUN_ANY_IN_BACKGROUND ignore to observe app behavior under power-saving conditions.
- Continuously test and adjust background behavior to provide a smooth user experience while adhering to Android's power-saving guidelines.

These strategies and best practices help developers navigate Android's power restrictions, optimizing both app functionality and battery efficiency for a balanced and user-friendly experience.

7. Future Directions

As the Android ecosystem continues to evolve, power management remains a focal point, with future versions likely to introduce even stricter energy-saving measures. Developers will need to adapt their applications to meet these new standards. Research in AI-driven energy

management, among other areas, could open new pathways for extending battery life without compromising performance.

7.1. AI-Driven Energy Management Solutions

AI presents a promising avenue for optimizing power consumption. AI-based models can predict user needs for real-time updates, defer non-essential tasks, and identify optimal times for network access [20]. Advancing AI-driven power optimization could significantly minimize unnecessary background activities, enhancing battery life while preserving user experience.

7.1.1. Research Opportunities:

- Development of AI algorithms that dynamically adjust app behavior in real-time based on predictive models of user behavior and device state.
- Exploration of on-device machine learning models that operate independently from the cloud, predicting optimal power usage scenarios and proactively managing app performance.
- Investigation into AI-powered task prioritization algorithms that learn from user activity, allocating resources to critical tasks while conserving energy on lower-priority processes.

7.2. Cross-Platform Energy Optimization Frameworks

With developers increasingly building apps for both Android and iOS, a unified approach to power optimization is essential. Cross-platform frameworks that allow developers to write code once and implement energy-saving strategies on multiple platforms could simplify development and ensure consistency [23, 24]. Cross-platform tools that effectively manage energy efficiency without compromising performance on either system are a promising area of research.

7.2.1. Research Opportunities:

- Development of cross-platform libraries that standardize power management and background task scheduling across Android and iOS, promoting consistent energy efficiency.
- Exploration of power-saving APIs compatible with hybrid or cross-platform environments, such as React Native or Flutter.
- Harmonizing platform-specific features, like Android's Adaptive Battery and iOS's Background App Refresh, to ensure consistent energy management across devices.

7.3. Optimizing Power Usage in Emerging Technologies

The integration of technologies like 5G, augmented reality (AR), and the Internet of Things (IoT) in mobile devices presents new challenges in balancing performance and battery life. These technologies require intensive processing and high network usage, quickly draining battery life [21, 22]. Research is essential to find ways for these technologies to operate on mobile devices efficiently.

7.3.1. Research Opportunities:

- Optimizing 5G networks for mobile battery efficiency, focusing on balancing high data throughput with lower power consumption.
- Development of energy-efficient rendering techniques for AR applications that maintain real-time performance while conserving battery life.
- Investigation into how IoT-connected mobile apps can manage power resources effectively, maintaining connectivity and functionality without excessive power drain.

7.4. Energy-Efficient Mobile AI and Edge Computing

The growing use of AI models on mobile devices, especially for edge computing, calls for a reevaluation of energy consumption [25]. Research should focus on optimizing AI model execution within power-saving features like Doze Mode and App Standby, ensuring these powerful tools don't heavily drain device resources [26].

7.4.1. Research Opportunities:

- Development of lightweight AI models optimized for mobile devices to minimize battery impact.
- Exploration of energy-efficient edge computing frameworks that shift intensive processing to the cloud while optimizing on-device performance under low-power conditions.
- Investigation into AI-driven task optimization for mobile devices, ensuring background services powered by AI models operate with minimal energy consumption while delivering high performance.

7.5. Context-Aware Energy Management

Context-aware computing holds significant potential for improving energy efficiency. By leveraging environmental data (e.g., location, time, user movement), apps can intelligently manage background activity and battery usage. This adaptive approach allows apps to respond dynamically to user behavior and environmental conditions, enhancing both power efficiency and user experience.

7.5.1. Research Opportunities:

- Development of context-aware task scheduling, where background tasks are triggered based on user context, such as when the user is stationary or connected to a network.
- Exploration of environment-driven optimizations, enabling apps to adjust background activity based on factors like location (e.g., connecting to Wi-Fi when near a known network).
- Investigation of adaptive notification systems that deliver important messages only when the user is in an optimal state to receive them, such as when charging or actively engaged with the device.

The future of mobile app development will be shaped by a shift towards smarter, energy-efficient strategies that blend machine learning, AI-driven optimizations, and enhanced user engagement features. Emerging technologies like 5G, AR, IoT, and edge computing bring new power challenges, necessitating continued research. Collaboration among developers, device manufacturers, and the research community will be key to developing solutions that balance performance, power, and user satisfaction in future Android applications.

References

[1] Google, "Power management," Android Developers. Available: https://developer.android.com/about/versions/pie/power. [Accessed: 25-Sep-2024].

[2] A. Gupta, B. Suri, D. Sharma, S. Misra, and L. Fernandez-Sanz, "Code Smells Analysis for Android Applications and a Solution for Less Battery Consumption," Scientific Reports, vol. 14, no. 1, pp. 17683, 2024.

[3] E. Souza, E. M. R. Barreto, and R. de Freitas, "Optimizing Energy Consumption," Intelligent Systems Design and Applications: Industrial Applications, Volume 6, vol. 6, p. 1, 2024.

[4] A. Ruiz Nepomuceno, E. López Domínguez, S. Domínguez Isidro, M. A. Medina Nieto, A. Meneses-Viveros, and J. de la Calleja, "Software Architectures for Adaptive Mobile Learning Systems: A Systematic Literature Review," Applied Sciences, vol. 14, no. 11, p. 4540, 2024.

[5] Y. Awad, I. Hegazy, and E.-S. M. El-Horbaty, "Power-saving actionable recommendation system to minimize battery drainage in smartphones," International Journal of Information Technology, pp. 1–9, 2024.

[6] C. Marimuthu, S. Chimalakonda, and K. Chandrasekaran, "How do open source app developers perceive API changes related to Android battery optimization? An empirical study," Software: Practice and Experience, vol. 51, no. 4, pp. 691-710, 2021, doi: https://doi.org/10.1002/spe.2928.

[7] C. Groza, D.-C. Apostol, M. Marcu, and R. Bogdan, "A Developer-Oriented Framework for Assessing Power Consumption in Mobile Applications: Android Energy Smells Case Study," Sensors (Basel, Switzerland), vol. 24, no. 19, pp. 6469, 2024.

[8] Y. M. Awad, E.-S. M. El-Horabty, and I. Hegazy, "Proposed Methodology for Battery Aging and Drainage Mitigation," International Journal of Intelligent Computing and Information Sciences, Ain Shams University, Faculty of Computer and Information Science, 2024.

[9] S. He, Y. Liu, and H. Zhou, "Optimizing Smartphone Power Consumption through Dynamic Resolution Scaling," presented at the 21st Annual International Conference on Mobile Computing and Networking (MobiCom), Paris, France, 2015.

[10] M. H. Memon, M. Hunain, A. Khan, R. A. Shaikh, and I. Khan, "Power management for

Android platform by Set CPU," in 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, 2016, pp. 3953-3958.

[11] V. Hurbungs, Y. Beeharry, A. K. Calkee, and G. Ahotar, "An Energy Efficient Android Application," ADBU Journal of Engineering (AJET), vol. 4, pp. 1-8, 2016.

[12] A. Cañete, J.-M. Horcas, I. Ayala, and L. Fuentes, "Energy efficient adaptation engines for android applications," Information and Software Technology, vol. 118, p. 106220, 2020, doi: https://doi.org/10.1016/j.infsof.2019.106220.

[13] G. F. Welch, "A survey of power management techniques in mobile computing operating systems," SIGOPS Oper. Syst. Rev., vol. 29, no. 4, pp. 47-56, Oct. 1995, doi: https://doi.org/10.1145/219282.219293.

[14] A. Abdelmotalib and Z. Wu, "Power Management Techniques in Smartphones Operating Systems," International Journal of Computer Science Issues, vol. 9, no. 6, pp. 78-85, 2012.

[15] L. Zhang, B. Tiwana, R. P. Dick, Z. Qian, Z. M. Mao, Z. Wang, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in 2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), Scottsdale, AZ, USA, 2010, pp. 105-114.

[16] R. Rua and J. Saraiva, "A large-scale empirical study on mobile performance: energy, runtime and memory," Empirical Software Engineering, vol. 29, no. 1, p. 31, Dec. 2023, doi: https://doi.org/10.1007/s10664-023-10391-y.

[17] A. A. Bangash, K. Ali, and A. Hindle, "A black box technique to reduce energy consumption of Android apps," in Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results, Pittsburgh, PA, USA, 2022, pp. 1-5, doi: https://doi.org/10.1145/3510455.3512795.

[18] A. Biørn-Hansen, C. Rieger, T.-M. Grønli, T. A. Majchrzak, and G. Ghinea, "An empirical investigation of performance overhead in cross-platform mobile development frameworks," Empirical Software Engineering, vol. 25, no. 4, pp. 2997-3040, Jul. 2020, doi: https://doi.org/10.1007/s10664-020-09827-6.

[19] A. Cañete, J.-M. Horcas, I. Ayala, and L. Fuentes, "Energy efficient adaptation engines for android applications," Information and Software Technology, vol. 118, p. 106220, 2020, doi: https://doi.org/10.1016/j.infsof.2019.106220.

[20] S. Chowdhury, S. Di Nardo, A. Hindle, and Z. M. Jiang, "An exploratory study on assessing the energy impact of logging on Android applications," Empirical Software Engineering, vol. 23, no. 3, pp. 1422-1456, Jun. 2018, doi: https://doi.org/10.1007/s10664-017-9545-x.

[21] M. Couto, P. Borba, J. Cunha, J. P. Fernandes, R. Pereira, and J. Saraiva, "Products go green: worst-case energy consumption in software product lines," in Proceedings of the 21st International Systems and Software Product Line Conference, vol. A, 2017, pp. 84–93.

[22] M. Couto, J. Cunha, J. P. Fernandes, R. Pereira, and J. Saraiva, "GreenDroid: A tool for analysing power consumption in the android ecosystem," 2015 IEEE 13th International Scientific Conference on Informatics, Poprad, Slovakia, 2015, pp. 73-78, doi: https://doi.org/10.1109/Informatics.2015.7377811.

[23] I. Zahid, M. A. Ali, and R. Nassr, "Android smartphone: Battery saving service," 2011 International Conference on Research and Innovation in Information Systems, 2011, pp. 1-4.

[24] A. M. Muharum, V. T. Joyejob, V. Hurbungs, and Y. Beeharry, "Enersave API: Android-based power-saving framework for mobile devices," Future Computing and Informatics Journal, vol. 2, no. 1, pp. 48-64, 2017.

[25] A. Almasri and L. B. Gouveia, "Analyzing and Evaluating the Amount of Power Consumption Used by Current Power-Saving-Applications on Android Smartphones," Internal Report TRS 04/2019. Technology, Networks and Society Group, 2019.

[26] N. Zaman and F. A. Almusalli, "Smartphones Power Consumption & Energy Saving Techniques," in 2017 International Conference on Innovations in Electrical Engineering and Computational Technologies (ICIEECT), 2017, pp. 1-7..