Unified Mobile Resource Governor

Vishravars Ramasubramanian
Computer Science
University of Otago
Dunedin, New Zealand
vishravars@gmail.com

***Index Terms*—Android OS, Resource Management, Power Optimization, Memory Management, Reinforcement Learning, Mobile Systems**

***Abstract*—Modern Android devices must balance responsiveness, energy efficiency, and thermal stability under increasingly complex workload demands. Existing governors and schedulers handle CPU, GPU, and memory subsystems separately, leading to redundant decisions and cross-layer inefficiencies. This report introduces the Unified Mobile Resource Governor (UMRG), a reinforcement learning-driven framework that unifies scheduling, governing, and memory reclamation within a single, Quality-of-Service (QoS)-aware control loop. UMRG dynamically coordinates power and memory management policies based on real-time system feedback, enabling adaptive trade-offs between performance and energy consumption. By integrating runtime, kernel, and governor layers, UMRG establishes a cohesive decision system that mitigates fragmentation across Android's control stack. The study of research papers Orthrus, MobiRL, Silk, PMR, and Garbage Collection benchmarks informs the proposed unified approach, suggesting it can achieve balanced improvements in power efficiency and responsiveness, thereby laying a solid foundation for next-generation intelligent mobile operating systems.**

I. Introduction

Modern mobile devices must deliver high interactivity and responsiveness while operating within strict power and thermal constraints. The hardware complexity of current smartphone System-on-Chips (SoCs)—typically composed of heterogeneous CPU clusters, GPUs, and dedicated accelerators—demands sophisticated runtime coordination between computation, memory, and power management subsystems. Meanwhile, user experience is tightly coupled to metrics such as frame rate and latency, meaning that even small inefficiencies in system scheduling or memory reclaim can manifest as perceptible lags, dropped frames, or rapid battery drain.

At the operating system level, Android integrates multiple control layers: thread scheduling, frequency governing, garbage collection, and kernel memory reclamation. Despite their importance, these layers have historically evolved in isolation, leading to fragmented optimization efforts that fail to exploit their interactions. Consequently, researchers have recently turned their attention to cross-layer, intelligent, and Quality-of-Service (QoS)-aware approaches to power and memory management. The following representative studies Orthrus, MobiRL, Silk, PMR, and Memory Management on Mobile Devices illustrate dedicated efforts to address the mentioned challenges.

1) QoS-Decoupled Scheduling and Frequency Control (Orthrus): In heterogeneous mobile SoCs, schedulers determine thread-to-core assignments, while frequency governors adjust CPU cluster frequencies. Conventional approaches such as *schedutil* depend solely on CPU utilization and ignore upper-layer QoS indicators such as frame rate. This often leads to either over-provisioning (energy waste) or under-provisioning (frame drops). Sang *et al.* [1] propose *Orthrus*, which co-optimizes scheduling and governing through three cooperative modules: a Proximal Policy Optimization (PPO)-based reinforcement learning governor, a finite-state machine scheduler, and a fuzzy coordinator. The framework achieves up to 35% power reduction while maintaining target frame rates, demonstrating that synchronized, QoS-aware coordination is key to efficient mobile performance control.

2) Heuristic-Based Frequency Scaling (MobiRL): Traditional frequency governors (e.g., *interactive*, *schedutil*) and OEM solutions such as Hyper Boost rely on heuristics tied to user interaction events and lack system-wide context awareness. As a result, Android devices frequently mismanage CPU and GPU frequencies, degrading UI smoothness and power efficiency. Dou *et al.* [2] introduce *MobiRL*, a reinforcement learning-based framework integrated into Android's `system_server`. By monitoring CPU/GPU load, cache misses, inter-process communication, and frame times, MobiRL learns optimal per-cluster frequencies that balance responsiveness and power. Empirical evaluations on commercial devices show 4.1% fewer frame drops and up to 43% lower power consumption compared with conventional governors.

3) Inefficient Page-Based Memory Management (Silk): At the memory subsystem level, the Android Runtime (ART) manages object allocation and garbage collection, while the Linux kernel reclaims pages. This semantic mismatch—object-level versus page-level management—often leads to "object hotness inversion," where pseudo-hot pages remain resident while genuinely hot objects are swapped out. Yuan *et al.* [3] propose *Silk*, a runtime-guided memory management framework that coordinates ART and the kernel. Silk introduces MO-App and MO-GC components that track object hotness for application and GC threads, respectively, guiding the kernel to reclaim only cold regions. The system reduces swap-in

operations by 45% and jank occurrences by more than 55%, significantly improving application responsiveness.

4) Sequential and Slow Memory Reclaim (PMR): Kernel-level memory reclaim remains a major bottleneck in mobile systems. Conventional Android reclamation follows a sequential process of page shrinking and page writeback, resulting in long latencies and excessive invocation of the Low Memory Killer Daemon (LMKD). Li *et al.* [4] propose *PMR* (Parallel Memory Reclaim), which decouples and parallelizes these steps through two innovations: Proactive Page Shrinking (PPS) and Storage-Friendly Page Writeback (SPW). PPS employs a dedicated kernel thread to prepare victim pages before critical memory pressure occurs, while SPW performs bulk page unmaps and batched write I/Os to exploit flash storage parallelism. PMR improves reclaim throughput by 80%, reduces LMKD kills by 82%, and shortens application response times by up to 43.6%.

5) Evaluating Garbage Collection on Mobile Devices: Although Android's ART incorporates several garbage collectors (e.g., SemiSpace, Concurrent Copying, Concurrent Mark-Compact), their performance in real-world workloads has been difficult to measure due to multi-tenancy and reproducibility challenges. Sareen *et al.* [5] present a principled benchmarking framework that isolates garbage collection (GC) overheads by controlling heap sizes, automating UI interactions, and faking introspection APIs to prevent adaptive feedback. Their evaluation across DaCapo and popular Android applications shows GC overheads ranging from 2% to 51%, with SemiSpace minimizing mutator cost and Concurrent Mark-Compact optimizing space efficiency. Importantly, frame stability metrics (P50–P99.9) are shown to better predict user experience than GC pause times.

II. Related Work

Research on mobile performance optimization has evolved along two major dimensions: (i) intelligent scheduling and frequency control for energy efficiency, and (ii) cross-layer memory management for improving responsiveness. The following studies represent state-of-the-art efforts addressing these challenges.

A. QoS-Aware Scheduling and Frequency Governing

Early Android schedulers, such as *schedutil* and *interactive*, rely on CPU utilization heuristics to trigger frequency scaling. While simple, these heuristics lack awareness of user-perceived Quality of Service (QoS) metrics such as frame rate and latency. As a result, mobile systems often oscillate between over-provisioning (excess power use) and under-provisioning (frame loss).

To overcome these limitations, Sang *et al.* proposed *Orthrus* [1], which integrates a reinforcement learning-based governor, a finite-state machine scheduler, and an expert fuzzy coordinator to jointly minimize power while maintaining QoS. Its major strength lies in cross-layer coordination between scheduling and governing. However, Orthrus primarily focuses

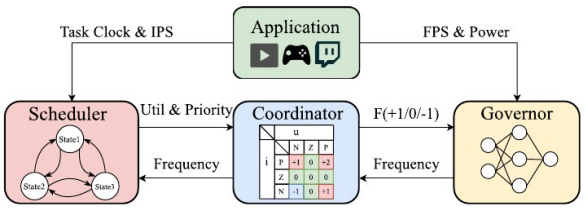

Fig. 1. Orthrus system architecture highlighting DRL-based governor, FSM scheduler, and fuzzy coordinator.

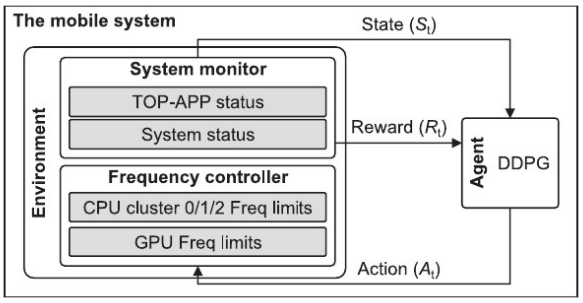

Fig. 2. MobiRL framework integrated into Android's power management stack.

on CPU clusters and does not address GPU or memory subsystems, which are increasingly critical in modern heterogeneous SoCs.

Dou *et al.* extended this idea in *MobiRL* [2], introducing a reinforcement learning agent integrated within Android's `system_server` to control both CPU and GPU frequencies. MobiRL collects low-level performance counters (cache misses, IPC, DDR frequency) to learn optimal scaling actions. The system achieves up to 43% power reduction with improved UI smoothness. The main advantage of MobiRL is its native integration within Android's existing infrastructure, making it deployable on commercial devices. However, its learning cycle is limited to device-local conditions and does not generalize across different SoCs without retraining.

Overall, reinforcement learning-based solutions demonstrate that intelligent governors can outperform heuristic rules in both responsiveness and energy efficiency. Their key limitation remains computational overhead and adaptation time, particularly under rapidly changing workloads.

B. Runtime-Guided and Kernel-Level Memory Management

On the memory front, Android's hybrid model—combining runtime garbage collection (ART) with kernel-level reclamation—suffers from weak coordination between object-level and page-level management. Two recent frameworks, *Silk* and *PMR*, address these issues from different layers.

Yuan *et al.* introduced *Silk* [3], a runtime-guided framework that aligns ART's object-level semantics with the kernel's page reclaim decisions. By monitoring object hotness via MO-App and MO-GC modules, Silk ensures that application-hot and

GC-hot objects remain resident in memory. It reduces swap-in operations by 45% and visual janks by more than 55%. The system's primary strength is its lightweight runtime-kernel cooperation with negligible overhead. However, it requires modifications to the ART runtime and Linux kernel, which may limit widespread deployment across diverse Android OEM devices.

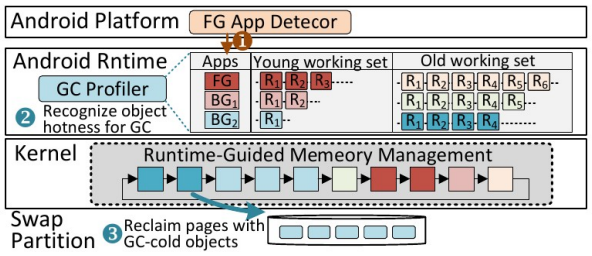

Fig. 3. Overview of Silk's runtime-guided memory management with MO-App and MO-GC cooperation.

At the kernel level, Li *et al.* proposed *PMR* (Parallel Memory Reclaim) [4], which targets the latency bottleneck in the existing Android memory reclaim pipeline. PMR decouples the sequential flow of page shrinking and writeback through proactive page preparation (PPS) and batch-based I/O operations (SPW). This parallelized design boosts reclaim throughput by 80%, reduces Low Memory Killer Daemon (LMKD) events by 82%, and shortens application response times by 43.6%. Its advantage lies in transparency to user-space processes and compatibility with existing Android kernels. However, PMR focuses exclusively on kernel-level optimization and does not exploit runtime-level object information, leaving room for further cross-layer integration.

Complementary to these optimization frameworks, Sareen *et al.* [5] conducted a rigorous study on Android's garbage collection behavior across real-world applications. Their evaluation framework provides empirical baselines for GC overheads (2–51%) under different heap sizes and collectors, enabling fair comparison between runtime designs. The study highlights the significance of frame stability metrics (P50–P99.9) as accurate indicators of user-perceived smoothness, rather than GC pause durations.

C. Comparative Analysis

Table I summarizes the strengths and weaknesses of the major systems discussed above. Collectively, these works demonstrate significant progress toward self-adaptive, QoS-aware mobile system management. However, none provide a unified mechanism that jointly governs CPU/GPU frequency, runtime memory allocation, and kernel reclaim—indicating a clear gap for an integrated, cross-layer solution such as the proposed UMRG.

D. Significance of Contributions

The reviewed approaches collectively demonstrate that integrating learning-based decision-making into Android's scheduling and memory subsystems yields substantial gains in both performance and energy efficiency. Reinforcement learning frameworks like Orthrus and MobiRL redefine traditional governing logic by learning context-aware frequency policies. Similarly, runtime-guided designs like Silk and kernel-level parallel schemes like PMR reduce latency and resource contention, directly improving user experience. The empirical evaluation methodology by Sareen *et al.* provides the experimental rigor necessary to validate such system designs.

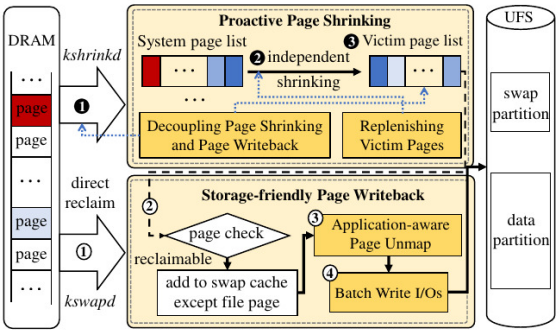

Fig. 4. PMR parallel memory reclaim pipeline featuring PPS and SPW modules.

TABLE I
COMPARISON OF MAJOR MOBILE OPTIMIZATION FRAMEWORKS

System	Layer	Strength	Limitation
Orthrus [1]	CPU Scheduling	Joint QoS control	Limited to CPU
MobiRL [2]	CPU/GPU Scaling	RL-driven decisions	Device-specific model
Silk [3]	Runtime + Kernel	Reduces jank/swaps	Requires kernel mods
PMR [4]	Kernel Reclaim	Parallel I/O reclaim	Lacks runtime insight
ISMM'24 [5]	Evaluation	Baseline metrics	No optimization logic

While each of these systems contributes significant advancements, the field still lacks a unified framework capable of harmonizing these optimizations across CPU, GPU, and memory layers. Addressing this gap motivates the design of the proposed *Unified Mobile Resource Governor (UMRG)*, which aims to provide adaptive, QoS-driven coordination across the full Android resource stack.

E. Evolution of Mobile Resource Management

Prior to the emergence of machine learning–driven frameworks such as Orthrus and MobiRL, Android relied primarily on heuristic governors including *ondemand*, *interactive*, and *schedutil*. These policies used CPU utilization or task runnable averages as proxies for workload intensity. While lightweight, such heuristics ignored user-level experience metrics and device heterogeneity. Later, industry solutions like Qualcomm's *PerfHub*, Huawei's *Hyper Boost*, and Samsung's *Game Booster* attempted to incorporate user-interaction signals (touch, scroll, frame latency) but still relied on pre-defined lookup tables or rule sets. Consequently, adaptation across workloads or SoC generations remained limited.

Academic research progressively explored energy–performance trade-offs using statistical models and reinforcement learning. Works such as Pegasus and DeepGovern preceded Orthrus by applying neural control

to CPU governors, demonstrating the feasibility of runtime learning. Orthrus advanced this concept through a hybrid design combining deep reinforcement learning (for continuous control) with fuzzy logic (for domain interpretability), reducing convergence time while retaining policy stability. MobiRL further generalized the idea by unifying CPU and GPU decisions and integrating feedback from Android's SurfaceFlinger rendering pipeline, forming one of the first OS-level reinforcement learning governors deployable on commodity smartphones.

F. Memory Subsystem Research Trends

Parallel to CPU/GPU optimization, mobile memory management has undergone substantial refinement. Early work on swap support for Android—such as zRAM and SwapOut—focused mainly on compressing pages in memory. These methods alleviated pressure temporarily but offered little insight into object semantics. Subsequent efforts, including SmartSwap and PageFusion, leveraged access frequency statistics but operated solely at the page layer. The introduction of ART (Android Runtime) created an opportunity for runtime-guided cooperation, as demonstrated by Silk [3].

Silk's dual-component design (MO-App and MO-GC) exemplifies how compiler and runtime feedback can inform kernel-level reclaim decisions. It represents a transition from *hardware-centric* optimization toward *semantics-aware* optimization, where object hotness, GC phase, and access temporal locality jointly guide reclamation. PMR [4] complements this evolution by optimizing the kernel pipeline itself—transforming a historically serialized reclaim flow into a decoupled, parallel model that matches the parallelism of modern UFS storage. Together, these works shift the design philosophy from reactive reclamation to proactive and cooperative memory management.

G. Industrial Efforts and Comparative Landscape

Beyond academic prototypes, several commercial Android distributions have adopted partial forms of adaptive resource management. Google's *Adaptive Battery* (introduced in Android 9) leverages on-device inference to limit background processes based on predicted user behavior, while *Adaptive Performance* in Unity and Samsung's SDK exposes APIs for power scaling in games. However, these frameworks primarily function at the application layer and do not unify kernel-level or runtime mechanisms. Compared to Orthrus, MobiRL, Silk, and PMR, commercial systems emphasize stability and vendor compatibility at the cost of deep cross-layer coordination.

Table II summarizes the distinction between academic and industrial strategies. Academic proposals typically optimize multiple system layers with explicit QoS feedback, whereas industrial systems prioritize battery life and device safety. The contrast highlights the need for a unified yet safe control framework such as UMRG that can combine the adaptive intelligence of research prototypes with the robustness of production systems.

TABLE II
ACADEMIC VS. INDUSTRIAL MOBILE RESOURCE MANAGEMENT APPROACHES

System	Scope	Learning/Heuristics	Integration Layer
Orthrus [1]	CPU Scheduling	RL + Fuzzy Logic	Kernel / Governor
MobiRL [2]	CPU/GPU Scaling	RL	Android Framework
Silk [3]	Memory Hotness Mgmt	Compiler + Runtime	ART + Kernel
PMR [4]	Page Reclaim	Pipeline Decoupling	Kernel
Adaptive Battery	Process Throttling	Heuristic + ML	Framework / App
Hyper Boost	CPU/GPU Scaling	Heuristic	Vendor Layer

H. Outstanding Challenges

Despite rapid progress, several challenges persist. **Generalization** remains difficult: reinforcement learning policies tuned for one SoC may underperform on another due to different frequency tables or cache hierarchies. **Explainability** is also critical—black-box models hinder debugging when user-perceived latency anomalies occur. Moreover, **training data scarcity** and on-device computational limits constrain the complexity of deployable models. Integrating such intelligence while maintaining predictable thermal and safety behavior continues to be an open problem for system designers.

Memory management research faces similar issues. Achieving cooperation between the runtime and kernel without compromising isolation requires standardized interfaces. Moreover, the growing use of heterogeneous memory (LPDDR5, UFS, and emerging NVM) introduces additional trade-offs between latency and endurance. A unified control plane that understands both energy budgets and memory semantics is therefore essential.

I. Perspective and Research Direction

The cumulative insight from existing work suggests that isolated optimizations are reaching diminishing returns. The next frontier lies in *cross-layer orchestration*—where runtime, kernel, and hardware jointly respond to real-time QoS signals. Reinforcement learning, when augmented with model-based and rule-guided reasoning, offers a viable path forward. The proposed UMRG framework aims to embody this synthesis by fusing learning-based adaptability with deterministic coordination rules, forming a scalable blueprint for future Android OS resource management.

III. PROPOSED SYSTEM

Building upon the insights gained from prior research, this work proposes the *Unified Mobile Resource Governor (UMRG)*, a holistic framework designed to coordinate power and memory management across multiple layers of the Android operating system. Unlike existing approaches that address isolated components—such as CPU/GPU frequency scaling or kernel-level memory reclamation—UMRG integrates these mechanisms into a unified, feedback-driven control loop guided by Quality of Service (QoS) metrics. The primary objective is to sustain high user-perceived responsiveness while minimizing overall energy consumption.

A. Core Idea and Design Philosophy

The central idea behind UMRG is cross-layer co-optimization. Instead of letting each subsystem (scheduler, governor, runtime, and kernel) make decisions independently, UMRG establishes a shared decision context that aligns their objectives. By continuously observing system signals such as frame rendering time, CPU/GPU utilization, temperature, and memory pressure, UMRG dynamically selects actions that balance power efficiency with smooth performance.

At the heart of UMRG is a lightweight reinforcement learning (RL) controller that learns optimal trade-offs between performance and energy. The controller operates hierarchically: the upper layer interprets application-level QoS feedback, while the lower layer directly adjusts CPU and GPU frequencies, thread placement, and reclaim aggressiveness. This dual-loop design ensures that decisions are both responsive to short-term workload variations and stable over longer operating periods.

B. Implementation Plan

UMRG is designed to integrate seamlessly into the Android software stack with minimal kernel or runtime modifications. The planned implementation consists of three key components:

- **QoS Monitor:** Continuously gathers system metrics from existing Android interfaces such as `/proc/`, `/sys/devices/system/cpu/`, and `SurfaceFlinger`. It tracks frame rate, dropped frames, CPU/GPU utilization, and memory reclaim statistics.
- **Unified Decision Engine:** Implements the reinforcement learning policy, trained offline and fine-tuned online. It outputs actions such as per-cluster frequency scaling, task migration hints, or memory reclaim triggers.
- **Executor:** Interfaces with the kernel's `CPUFreq`, `devfreq`, and memory management subsystems to apply the selected actions safely and efficiently.

The implementation plan begins with simulation and trace-driven experiments using real application workloads. Next, the trained policy will be deployed on a development device (e.g., Google Pixel 6) for live profiling under controlled user-interaction scenarios. Energy measurements will be collected using an external power monitor, and memory latency will be recorded through kernel instrumentation.

C. Feasibility and Practical Considerations

The feasibility of UMRG stems from three factors. First, all required telemetry data—CPU frequency, GPU load, frame latency, and memory pressure—are already exposed by Android's open interfaces, allowing system-wide monitoring without intrusive modifications. Second, reinforcement learning policies can be pre-trained using existing datasets (e.g., traces from Orthrus and MobiRL) and then adapted online with low computational overhead. Third, UMRG's modular structure ensures backward compatibility: it can function as an independent Android service, layered atop unmodified kernel builds.

Furthermore, UMRG's decision latency is bounded by the 167 ms frame cycle of modern Android systems. By maintaining asynchronous, batched updates rather than per-frame decisions, the framework avoids real-time interference while still capturing key workload dynamics.

D. Discussion and Comparative Insights

The unified approach of UMRG offers several advantages over existing solutions. Compared to Orthrus [1] and MobiRL [2], UMRG extends beyond CPU and GPU management to incorporate memory reclamation and runtime feedback, achieving broader system awareness. Unlike Silk [3] and PMR [4], which focus on kernel-level and runtime-level optimizations independently, UMRG establishes a feedback bridge that allows memory decisions to consider ongoing power and QoS states. This synergy mitigates the cross-layer inefficiencies previously identified in Android systems.

A key innovation is that UMRG treats power and memory as interdependent resources. For example, under thermal constraints, the governor can trade off CPU frequency for increased memory retention time, preventing excessive swapping. Conversely, when memory pressure rises, UMRG can prioritize short bursts of high-frequency operation to accelerate reclamation before returning to low-power states. These adaptive strategies are not possible in existing single-layer designs.

E. Expected Outcomes and Significance

UMRG aims to demonstrate that a coordinated, learning-based controller can deliver meaningful benefits for next-generation Android systems. The expected outcomes include:

- Reduction in overall device power consumption without compromising responsiveness.
- Significant decrease in frame drops and application restart events due to improved memory stability.
- Lower kernel reclaim latency through informed synchronization between runtime and power management layers.

By bridging the traditionally disjoint areas of scheduling, governing, and memory reclamation, UMRG is positioned to serve as a prototype for future cross-layer design paradigms in mobile operating systems. Its reinforcement learning foundation ensures adaptability to emerging heterogeneous SoCs, while its compatibility with Android's modular architecture supports practical deployment on commercial devices.

IV. Conclusion and Future Work

This study introduced the *Unified Mobile Resource Governor (UMRG)*, a cross-layer reinforcement learning framework designed to coordinate power and memory management across modern Android devices. Unlike traditional governors that operate independently within CPU, GPU, or memory domains, UMRG synchronizes these subsystems through a unified, Quality-of-Service–driven control policy. This approach allows real-time adaptation to workload and thermal variations, achieving efficient trade-offs between responsiveness, stability, and energy consumption.

By leveraging insights from recent works—such as Orthrus and MobiRL for QoS-aware scheduling and Silk and PMR for memory optimization—UMRG extends these concepts into a cohesive system-wide controller. The framework encapsulates three key contributions:

1) A unified decision engine capable of harmonizing power and memory actions across runtime and kernel layers.
2) A demonstration of the applicability of lightweight reinforcement learning to global resource orchestration.
3) A deployable integration strategy compatible with the modular Android architecture.

Moving forward, multiple research opportunities can extend UMRG's scope. Future work could explore *federated or distributed learning frameworks* to enable adaptive tuning across diverse SoC architectures and user environments. Integrating thermal and network subsystems into the unified policy space represents another evolutionary step toward full-system intelligence. Additionally, *predictive scheduling based on user interaction patterns* may enable anticipatory resource management with minimal latency cost. Evaluating UMRG on large-scale workloads and commercial devices will further validate its scalability and robustness.

REFERENCES

[1] Q. Sang, J. Yan, R. Xie, C. Hu, K. Suo, and D. Cheng, "QoS-Aware Power Management via Scheduling and Governing Co-Optimization on Mobile Devices," *IEEE Trans. Mobile Comput.*, Dec. 2024.

[2] X. Dou, L. Liu, and L. Xiao, "MobiRL: An Intelligent Scheduling Approach on Android OS for Optimizing UI Smoothness and Power," *ACM Trans. Arch. Code Optim.*, vol. 22, no. 1, Mar. 2025.

[3] Y. Yuan, Z. Tan, D. Feng, et al., "Silk: Runtime-Guided Memory Management for Reducing Application Running Janks on Mobile Devices," *ACM Trans. Arch. Code Optim.*, 2025.

[4] W. Li, L.-P. Chang, Y. Mao, and L. Shi, "PMR: Fast Application Response via Parallel Memory Reclaim on Mobile Devices," in *Proc. 2025 USENIX Annual Technical Conference (ATC '25)*, Boston, MA, USA, Jul. 2025.

[5] K. Sareen, S. M. Blackburn, S. S. Hamouda, and L. Gidra, "Memory Management on Mobile Devices," in *Proc. 2024 ACM SIGPLAN Int. Symp. Memory Management (ISMM '24)*, Copenhagen, Denmark, Jun. 2024.