KMAN PUBLISHING INSTITUTE

Article history:
Received 03 September 2025
Revised 03 December 2025
Accepted 06 January 2025
Published online 01 September 2026

Journal of Resource Management and Decision Engineering

Volume 5, Issue 3, pp 1-13

Design and Optimization of a Real-Time Audio Simulation Engine on Mobile Devices

Mahdi. Habibi[1*], Amirhossein. Mirahmadi[2], Mohammad Mahdi. Jalili[3]

[1] B.A., Department of Management, Karaj Sugar Production Company University of Applied Science and Technology, Karaj, Iran
[2] M.A., Department of Computer Engineering, Shahid Bahonar University, Kerman, Iran
[3] B.A., Department of Computer Engineering, Faculty of Computer Engineering, Hamadan University of Technology, Hamadan, Iran

*** Corresponding author email address**: themahdihabibi@gmail.com

A r t i c l e I n f o

Article type:
Original Research

How to cite this article:
Habibi, M., Mirahmadi, A. & Jalili, M. M. (2026). Design and Optimization of a Real-Time Audio Simulation Engine on Mobile Devices. *Journal of Resource Management and Decision Engineering*, 5(3), 1-13.
https://doi.org/10.61838/kman.jrmde.5.3.237

A B S T R A C T

With the rapid expansion of interactive and multimedia applications on smartphones, real-time audio simulation has become one of the core components in user experience design. However, the inherent limitations of mobile platforms in terms of computational capacity, energy consumption, and strict real-time constraints have turned the design of stable and low-latency audio engines into a major technical challenge. The objective of the present study is to design and optimize an efficient architecture for a real-time audio simulation engine on mobile devices that can establish an appropriate balance between audio quality, real-time responsiveness, and resource consumption. This study was conducted using a design-oriented and experimental approach. First, a system-centered architecture based on the separation of real-time and non-real-time domains was developed. Subsequently, a set of optimization algorithms and techniques—including adaptive buffer management, voice capping and voice stealing policies, quality scaling, and conditional processing—were implemented. The proposed engine was developed on the Android platform using low-level audio APIs and evaluated through an interactive case study. The system's performance was compared with that of a baseline implementation. The experimental results demonstrated that the proposed architecture significantly reduced latency and jitter while maintaining the real-time stability of the engine under high-load conditions. In addition, CPU usage and energy consumption were reduced in a controlled manner, and the degradation of audio quality was applied gradually and in a manner perceptually acceptable to users. Perceptual findings further indicated that users perceived controlled quality degradation as considerably more tolerable than audio instability or dropouts. The findings suggest that the design of real-time audio simulation engines on mobile platforms should be grounded in an architectural and adaptive approach. Emphasizing real-time pipeline management and intelligent control policies plays a more decisive role in achieving stable and efficient performance than increasing the complexity of digital signal processing (DSP) algorithms.

Keywords: *Real-Time Audio Processing; Mobile Audio Engine; Latency and Jitter; Adaptive Optimization; Interactive Multimedia Systems*

1. Introduction

The contemporary digital ecosystem is increasingly shaped by immersive, interactive, and multimedia-rich applications, in which audio has emerged as a decisive factor in shaping user experience, engagement, and perceived realism. From mobile gaming and augmented reality (AR) to social media, virtual collaboration platforms, and the emerging audio metaverse, sound is no longer a secondary enhancement but a core functional component of system design (Collins, 2008; Jot et al., 2021; Sweet, 2014; Yang et al., 2022). Recent developments in human–computer interaction emphasize that users' perception of system responsiveness, spatial presence, emotional engagement, and overall satisfaction are strongly mediated by the quality, stability, and real-time behavior of audio rendering pipelines (Fırat et al., 2022; Xue & Zheng, 2025; Yang et al., 2022). Consequently, the engineering of real-time audio systems has become a strategic concern for designers of mobile and interactive platforms.

Mobile computing now dominates global digital interaction, yet mobile platforms impose severe constraints on computational resources, energy consumption, memory capacity, and timing determinism. These constraints directly challenge the feasibility of high-quality real-time audio simulation. The coexistence of heterogeneous hardware architectures, dynamic operating system scheduling, aggressive power management policies, and competing application workloads introduces nontrivial sources of latency, jitter, and instability in audio pipelines (Jahangashteh et al., 2022; Ota et al., 2017; Zhao et al., 2022). While desktop-class systems can often tolerate inefficient designs through brute-force computing power, mobile systems demand carefully engineered architectures that explicitly balance performance, resource efficiency, and real-time reliability.

At the same time, the functional role of audio has expanded dramatically. Modern applications increasingly rely on complex audio behaviors such as spatial sound rendering, adaptive soundscapes, interactive music systems, and procedural audio synthesis. These behaviors require continuous parameter modulation, low-latency response to user input, and seamless integration of multiple concurrent sound sources (Farnell, 2010; Lazzarini et al., 2016; Pulkki & Karjalainen, 2015). The growth of audio augmented reality and spatial audio for immersive environments further intensifies the computational burden placed on mobile audio engines (Fırat et al., 2022; Jot et al., 2021; Yang et al., 2022). These trends have elevated the design of real-time audio engines from a purely technical problem to a strategic design challenge that directly affects product success and user adoption.

Concurrently, advances in machine learning and large-scale audio modeling have begun to influence audio processing pipelines. Deep learning has enabled powerful new capabilities such as content-aware sound synthesis, intelligent noise suppression, adaptive sound classification, and perceptual optimization (Deng, 2019; Latif et al., 2023; Zhao et al., 2022). However, deploying such techniques on mobile devices introduces new constraints, as the computational and energy costs of deep neural models must be carefully managed in real time (Ota et al., 2017; Zhao et al., 2022). The integration of intelligent audio processing therefore magnifies the importance of architectural efficiency, adaptive resource management, and stable real-time execution.

Despite the central importance of audio, many mobile applications still rely on simplistic audio frameworks that were not designed for heavy interactive workloads. Traditional mobile audio APIs provide basic playback and mixing services but often lack advanced scheduling control, fine-grained resource management, and robust mechanisms for ensuring deterministic timing under load (Jahangashteh et al., 2022; Schobel et al., 2016). As application complexity grows, these limitations become increasingly visible to users through audible artifacts such as dropouts, clicks, inconsistent spatialization, delayed feedback, and unstable audio scenes. Empirical studies in interactive media demonstrate that such artifacts significantly degrade perceived quality and user engagement, even when visual performance remains high (Collins, 2008; Fırat et al., 2022; Sweet, 2014).

The management implications of these technical challenges are substantial. For firms operating in competitive digital markets, the quality and reliability of interactive audio systems directly influence customer retention, brand perception, and market differentiation. Audio failures in games, AR applications, or collaborative platforms can erode trust and reduce user adoption, while stable and immersive sound design can enhance perceived innovation and product value (Khan, 2024; Xue & Zheng, 2025). Consequently, the design of real-time audio engines

is no longer confined to engineering departments but has become a strategic asset that shapes product competitiveness and organizational performance.

From a system design perspective, the core difficulty lies in maintaining strict real-time constraints while simultaneously executing complex audio processing, resource management, and application logic on limited mobile hardware. Real-time audio threads must complete their computations within hard deadlines imposed by buffer scheduling; failure to do so results in underruns and audible dropouts. Meanwhile, non-real-time application components—such as user interfaces, networking, file I/O, and analytics—compete for the same underlying resources, creating unpredictable execution patterns (Jahangashteh et al., 2022; Schobel et al., 2016). The absence of robust architectural separation between these domains remains a fundamental weakness of many existing systems.

Prior research in audio engineering, computer music, and multimedia systems has established numerous foundational principles for real-time sound synthesis, signal processing, and spatial audio rendering (Farnell, 2010; Lazzarini et al., 2016; Pulkki & Karjalainen, 2015). However, much of this literature was developed in contexts where computational resources were relatively abundant or where systems operated under controlled laboratory conditions. The translation of these principles into mobile environments—characterized by energy constraints, heterogeneous hardware, and unpredictable operating system behavior—requires significant architectural adaptation (Ota et al., 2017; Zhao et al., 2022).

Recent surveys of mobile multimedia and deep learning on mobile devices highlight that efficient system-level optimization, rather than raw algorithmic sophistication, often determines practical success (Deng, 2019; Ota et al., 2017; Zhao et al., 2022). This observation aligns with emerging work on lightweight mobile process engines and adaptive system architectures, which emphasizes the importance of minimizing critical execution paths, isolating time-sensitive workloads, and applying dynamic quality control to preserve responsiveness under fluctuating resource conditions (Schobel et al., 2016; Xue & Zheng, 2025). In the audio domain, this suggests that architectural design and resource governance may play a more decisive role in achieving stable performance than incremental improvements in signal processing algorithms alone.

Moreover, the evolution of immersive media ecosystems—particularly AR, VR, and the audio metaverse—has introduced new requirements for interoperability, spatial coherence, and perceptual consistency across devices and platforms (Fırat et al., 2022; Jot et al., 2021; Yang et al., 2022). These requirements further complicate mobile audio engine design by demanding scalability, adaptability, and cross-platform compatibility, all within the strict confines of mobile hardware constraints.

Despite these developments, the literature reveals a notable gap: while substantial research addresses individual components of audio processing, spatialization, or mobile optimization, comparatively little work has proposed integrated architectural frameworks that holistically address latency, jitter, resource efficiency, and perceptual quality in real-time mobile audio engines. Existing studies often focus either on high-level interaction design (Collins, 2008; Sweet, 2014), low-level signal processing (Farnell, 2010; Pulkki & Karjalainen, 2015), or mobile system optimization (Ota et al., 2017; Zhao et al., 2022), without unifying these perspectives into a coherent real-time engine architecture.

In addition, as audio content becomes increasingly procedural, adaptive, and data-driven, the complexity of real-time control systems grows accordingly. Advanced audio behaviors now depend on dynamic parameter scheduling, real-time event handling, and continuous environmental feedback, all of which impose new constraints on engine stability and performance (Jot et al., 2021; Latif et al., 2023; Yang et al., 2022). Without systematic architectural support, these features risk overwhelming mobile platforms and undermining user experience.

From a managerial viewpoint, the lack of robust architectural solutions introduces operational risks. Development teams may resort to ad hoc optimizations that yield short-term improvements but lack scalability and maintainability. Such approaches increase technical debt, prolong development cycles, and complicate future product evolution. In contrast, well-designed audio engine architectures can serve as reusable organizational assets, reducing development cost, accelerating innovation, and supporting long-term product strategies (Khan, 2024; Xue & Zheng, 2025).

Taken together, these considerations underscore the urgent need for systematic, architecture-driven approaches to the design of real-time audio simulation engines for mobile platforms. Such approaches must explicitly address latency and jitter control, adaptive resource management, scalability under dynamic workloads, and preservation of perceptual quality, while remaining compatible with the

practical constraints of mobile hardware and operating systems (Ota et al., 2017; Schobel et al., 2016; Xue & Zheng, 2025; Zhao et al., 2022).

Accordingly, this study is situated at the intersection of interactive media engineering, mobile computing, and management-oriented system design. By integrating insights from audio engineering, multimedia systems, and mobile optimization research, it seeks to contribute both technically and strategically to the development of more reliable and efficient mobile audio infrastructures (Farnell, 2010; Khan, 2024; Lazzarini et al., 2016; Pulkki & Karjalainen, 2015; Zhao et al., 2022).

The aim of this study is to design and empirically evaluate an adaptive, architecture-driven real-time audio simulation engine for mobile platforms that minimizes latency and jitter while optimizing resource consumption and preserving perceptual audio quality under dynamic workload conditions.

2. Methods and Materials

Design of the Audio Simulation Engine Architecture

The proposed architecture for a real-time audio simulation engine on mobile devices is presented; an architecture whose objective is to reduce latency and jitter, ensure output stability, and optimize CPU, memory, and energy consumption under real-world operating conditions. The central design concept is the construction of a short, predictable, and low-overhead processing path that is decoupled from application logic and executed within an independent audio loop. The primary objectives include real-time responsiveness, temporal stability, lightweight operation, and controlled scalability.

Mobile Platform Constraints

Mobile platforms impose strict constraints, including limited CPU and GPU resources, operating system energy management policies, thread scheduling restrictions, and substantial heterogeneity in hardware and audio APIs. The architecture is therefore designed as a layered pipeline comprising two distinct execution domains:

Audio Thread

A real-time loop responsible exclusively for time-critical operations:

Mixing → DSP → Output Buffer

Control Thread

Handles resource loading, audio scene management, analytics, input/output operations, networking, and user interface processing. This separation ensures that application-level interruptions and heavy workloads exert minimal influence on audio output stability.

Core System Modules

Audio I/O Backend

Interface with the operating system's audio output.

Android: AAudio (preferred) or OpenSL ES

iOS: Core Audio / Audio Units

Its function is to respond to output buffer fill requests and provide PCM frames.

Audio Graph / Routing

The engine is conceptually modeled as an audio graph:

Nodes: Source, Mixer, Effect, Spatializer, Output

Edges: Signal paths

This model enables structured composition of sources and effects without introducing complexity into the real-time loop.

Audio Resource Management

Management of concurrent channels and implementation of voice stealing policies. Prioritization is enforced (e.g., UI/interaction sounds supersede ambience), low-importance sources are attenuated or terminated under CPU pressure, and computational overload in dense scenes is prevented.

Event and Parameter System

Application events are transformed into audio commands (Play/Stop, parameter changes). Commands are transmitted to the Audio Thread using lock-free or minimally locked structures to minimize blocking latency.

DSP Engine

Signal processing chain:

Resampling

Filtering / Equalization

Dynamics / Compression

Lightweight or convolution-based reverb

Spatialization (Stereo Panning / HRTF)

The DSP design is block-oriented and dynamically enabled or disabled in response to system load.

Buffer Manager

Manages input/output buffers and prevents underruns:

Double/Triple buffering

Ring buffers for inter-thread data exchange

This module is essential for latency control while preserving system stability.

Resource and Streaming Manager (Non-Real-Time)

Handles file loading, decoding (AAC/Opus/MP3), caching, and streaming. The output is delivered as preprocessed buffers to the Audio Thread.

Data and Control Flow

Control flow: The application generates an event (e.g., "collision sound") → the event is placed in the Event Queue → the Audio Thread retrieves the queue at the start of each callback and updates state without heavy processing.

Data flow: Voices generate or read samples → signals are summed in the Mixer → DSP is applied → the output buffer is filled and delivered to the Backend.

Concurrency and Threading Design

Minimum recommended configuration: one real-time thread, one control thread, and one optional auxiliary thread.

Core Principles

No memory allocation within callbacks, no file/network I/O, no heavy locks or long mutex operations, and strictly predictable execution time. Ring buffers are employed for continuous data streams, and lock-free or ultra-lightweight queues are used for control messages.

Key Design Decisions for Latency and Jitter Reduction

Critical path minimization: All non-essential operations are removed from the Audio Thread.

Adaptive buffer sizing: Under CPU pressure, the buffer is moderately increased to avoid underruns; under stable conditions, it is reduced to minimize latency.

Adaptive quality control: Computationally intensive effects are disabled or degraded when system load rises.

Precomputation and caching: Wavetables, filter coefficients, routing paths, and lookup tables are prepared in advance.

Voice capping and stealing: Upper bounds are imposed on concurrent voices, with intelligent removal policies.

To ensure implementability and testability, the engine's internal API is recommended to expose the following interfaces:

Engine.init

Engine.submitEvent

Engine.setParam

Engine.render

Engine.setQualityMode

Architectural Validity Criteria

The architecture is considered successful if callbacks consistently complete before their deadlines, jitter remains low and stable, performance degradation is gradual and controlled as voice count increases, and perceptual audio quality in real-world scenarios is preserved or degrades in a "justifiable and manageable" manner. The proposed architecture is founded on a single critical principle: strict separation of the real-time domain from the control and non-real-time domain. Any operation that is potentially time-consuming, unpredictable, or I/O-dependent is excluded from the audio execution path to minimize latency and jitter and guarantee system stability.

Primary Execution Domains

A) Audio Thread

This thread is invoked by the operating system (via callback or pull model) to fill the output buffer and executes only time-critical tasks: ingestion of lightweight control messages, generation or reading of active voice samples, multichannel mixing, lightweight and configurable DSP (EQ, filtering, spatialization, reverb), output buffer population, and delivery to the Audio Backend.

Strict Constraints: no memory allocation, no I/O, no heavy locking, no unpredictable operations.

B) Control / Application Thread (Non-Real-Time)

All operations that may induce blocking or timing variability are handled here: application and game logic, user interface processing, file loading, effect and parameter preparation, high-level decision making, logging, and system monitoring.

Core System Blocks

Operating system audio API bridge (AAudio/OpenSL ES on Android, Core Audio on iOS) responsible for buffer exchange and preservation of sample rate and format integrity.

The real-time core consists of:

Real-Time Engine Core

Event / Command Ingest

Voice Manager

Mixer

DSP Chain

Buffer Manager

Command Queue

A lightweight, preferably lock-free, command transmission path is established from the Control Thread to the Audio Thread for:

Play/Stop

SetParameter (gain, pitch, position, filter)

ChangeScene / Route

Resource and Streaming Subsystem (Non-Real-Time)

Handles loading, decoding, caching, and streaming of audio assets; outputs "consumption-ready buffers" for Voices.

Profiler and Monitor (Non-Real-Time)

Collects metrics including CPU load, underrun count, callback execution time, jitter, estimated latency, and adaptive quality decisions.

Overall Data and Control Flow

Control flow: Application generates an event → event is converted into an audio command → command enters the Command Queue → Audio Thread retrieves the command at callback start and updates state.

Data flow: Voices generate or read samples → Mixer combines output → DSP applies processing → Buffer Manager manages the buffer → Audio Backend transmits the buffer to hardware.

Conceptual Architecture Representation

Non-Real-Time Domain
App/UI/Game Logic
Resource Loader / Decoder / Streamer
Profiler and Quality Controller
↓ (Commands / Parameters)
Command Queue (RT-Safe)
↓
Real-Time Domain
Event Ingest
Voice Manager
Mixer
DSP Chain
Buffer Manager
↓
Audio Backend (AAudio / Core Audio)
↓
Speaker / Headphones

Stability and Optimization Logic in the Overall Architecture

Real-time loop remains short and predictable: only essential processing occurs within callbacks.

Quality scaling: if CPU load increases, heavy effects are degraded or disabled to prevent underruns.

Voice capping and stealing: concurrent voices are capped and low-priority sources are removed.

Adaptive buffering: buffer size is adjusted within permissible limits to balance latency and stability.

Resource pre-decoding and preparation outside the real-time domain to prevent jitter.

Optimization Algorithms and Methods

This section presents a set of implementable methods for optimizing the audio engine that directly affect latency, jitter, output stability, and CPU/memory/energy consumption. The overarching approach is to tightly control the "critical path" (Audio Thread) and migrate any non-deterministic or heavy operations to the non-real-time domain.

Latency Reduction at the Architectural and Buffer Levels

Real-time performance optimization in audio processing engines requires coordinated decisions at the architecture, scheduling, and resource-management levels, with a core emphasis on reducing delay and controlling temporal variability. One of the most fundamental elements in this pathway is buffer-size optimization because buffer latency is directly proportional to buffer size. Specifically, if the sampling rate is $f_s$and the buffer size is N, the buffer-induced delay is approximately $\text{Latency}_{buffer} \approx N/f_s$. Accordingly, the optimal strategy is to select the smallest stable buffer size and then, through continuous monitoring of underrun events, dynamically apply a bounded increase in buffer size under computational pressure—an approach commonly referred to as adaptive buffering.

Alongside buffer management, shortening the audio output path is of particular importance. In practice, unnecessary format conversions—such as repeated float-to-int and int-to-float casts—should be avoided within the Audio Thread. Likewise, resampling should be eliminated from the real-time path; if resampling is unavoidable, only lightweight, low-cost, preconfigured resamplers should be used to prevent unpredictable computational load.

From the perspective of perceived audio quality, smooth parameter scheduling is critical. Changes to gain, pitch, or filter parameters should not be applied abruptly; instead, linear or exponential ramps should be applied over multiple consecutive frames to prevent clicks and artifacts.

Jitter control and real-time stability also require RT-safe design, meaning that no non-deterministic operations occur within the Audio Thread. Concretely, the use of malloc/new, file or network I/O, heavy logging, complex string operations (e.g., regex processing or string formatting), and long-held mutex locks should be strictly prohibited. In contrast, only constant-time computations, direct array accesses, and lock-free queue structures should be permitted.

Consistent with this design logic, using a lock-free or minimally locked message queue for communication between the Control Thread and the Audio Thread is recommended. Commands such as Play, Stop, and SetParam should be generated on the Control Thread, while the Audio Thread should consume them in batches at the beginning of each callback.

In addition, file decoding and network streaming should be delegated to separate threads, and decoded output should be prepared as PCM chunks in a ring buffer, such that the Audio Thread is responsible solely for reading ready-to-consume data.

Finally, configuring the scheduling policy and priority of the audio thread—within the constraints imposed by the operating system—is highly consequential. A reasonable elevation of Audio Thread priority can reduce preemption and callback execution variability, thereby materially improving real-time stability and perceived audio quality.

Algorithm-Level DSP Optimization

Algorithm-level optimization of digital signal processing (DSP) is among the most effective strategies for achieving stable, low-latency performance in audio engines, particularly on mobile platforms where compute and energy constraints make algorithm design a critical challenge. In this context, block-based processing (as opposed to sample-by-sample processing) plays a central role in reducing computational overhead. Applying effects to blocks of audio frames improves memory locality and significantly reduces the overhead of repeated function calls.

Consistent with this approach, designing simple loops, using small and inline functions, and avoiding complex conditional and heavily branched execution paths can make execution time more predictable and reduce jitter on the Audio Thread.

Effect selection should also be guided by a mobile-first, lightweight mindset. For example, while full convolution reverb with long impulse responses can be perceptually superior, it is computationally expensive and typically unsuitable for most mobile use cases. Instead, lightweight structures such as Schroeder reverberators or feedback delay networks (FDNs) with a limited number of taps can offer a practical balance between quality and efficiency.

Similarly, in spatialization, many mobile scenarios are adequately served by simple stereo panning combined with distance attenuation. Full head-related transfer function (HRTF) processing should be restricted to high-quality modes or specific conditions to avoid unnecessary computational cost.

A subtle but operationally important challenge in floating-point DSP is the presence of denormal numbers—very small floating-point values that can cause severe CPU slowdowns. This issue commonly appears in filters and reverb tails and, if unaddressed, can compromise real-time stability. Common mitigation techniques include adding a very small, controlled noise floor to the signal or enabling flush-to-zero (FTZ) and denormals-are-zero (DAZ) modes on platforms and processors that support these features.

Finally, intelligent use of precomputation and lookup tables (LUTs) is among the most effective techniques for reducing real-time computational load. Expensive functions such as sine/cosine for LFOs, envelope curves, and constant filter coefficients should be precomputed and stored in tables rather than repeatedly evaluated within the Audio Thread. By removing repetitive high-cost operations from the real-time path, this approach improves throughput, enhances temporal stability, and supports overall DSP engine quality.

Low-Level Computational Optimization

Low-level computational optimization is a critical layer in real-time audio engine design that directly affects performance, energy usage, and temporal stability, particularly on mobile hardware. One of the most effective tools at this level is exploiting SIMD capabilities—especially NEON on ARM architectures and Android devices. SIMD enables parallel execution of repeated, independent operations such as signal mixing, gain application, short FIR filtering, and other vector computations across multiple audio samples simultaneously.

The greatest benefit is achieved in tight loops that process contiguous arrays of PCM samples, where parallelism can significantly reduce CPU cycles and increase execution-time predictability. However, effective SIMD usage requires careful data layout and appropriate memory alignment to avoid unintended overhead.

In addition to floating-point computation, selective use of fixed-point arithmetic in specific components can be an effective strategy for reducing CPU load, especially on lower-end devices or in effects with constrained dynamic range and simpler computational patterns. Although fixed-point can offer speed and energy advantages compared with float, it introduces nontrivial trade-offs: algorithm design, scaling, and calibration become more complex, and the risks of reduced numerical precision and clipping increase. Therefore, fixed-point should be used selectively and only after a careful analysis of hardware constraints and quality requirements.

At the level of data architecture and control flow, reducing branching and cache misses plays a decisive role in real-time performance. Excessive conditional branching can degrade branch prediction and introduce variability in callback execution time; thus, branchless designs or minimal conditional logic—particularly in the real-time path—are recommended.

Memory layout is equally important. Storing voice data and mixing parameters in cache-friendly structures can increase cache hit rates. In this context, adopting a structure-of-arrays (SoA) pattern rather than an array-of-structures (AoS) is often more efficient for mixing loops because it

facilitates sequential access to homogeneous arrays and reduces unnecessary data loads.

Finally, heavy polymorphism, virtual functions, and dynamic dispatch should be avoided in the real-time path because they increase branching and reduce execution-time predictability, which is inconsistent with the strict constraints of real-time audio processing.

Optimization of Audio Resource Management

Managing concurrent voices and computational resources in real-time audio engines becomes determinative when the number of audio events exceeds hardware capacity. Voice control strategies can therefore distinguish between stable output and a user experience dominated by underruns and quality degradation.

As a foundational policy, voice capping limits the maximum number of simultaneous voices based on device capability (e.g., 16, 24, or 32) to prevent saturation. However, because real-world demand may exceed this cap, the system must apply intelligent eviction strategies.

Priority- and perceptually informed voice stealing is crucial in this context. Lower-priority sounds are sacrificed before critical UI or high-salience effects (e.g., UI > SFX > ambience). Current signal amplitude is also a practical criterion: quieter voices are removed first to minimize perceptual disruption. Distance from the listener is another scene-based perceptual metric; more distant sources are typically removed at lower perceptual cost. Many systems additionally consider "time remaining to completion," as retaining a nearly finished sound can prevent abrupt and artificial cutoffs.

Beyond removal, virtualization is a more nuanced strategy for distant or low-importance sources. Instead of executing the full DSP chain, the engine updates only the logical state while keeping the actual output at or near zero. This preserves scene coherence (e.g., enabling natural re-entry when the listener approaches) while minimizing processing cost.

In parallel, memory optimization and eliminating allocations in the real-time path are essential. Using memory pools or object pools for entities such as voices, events, and buffers removes dynamic allocation from the Audio Thread and improves temporal predictability.

Zero-copy buffering is also valuable: by reducing multi-stage copying, it lowers CPU overhead and memory bandwidth pressure. The ideal data path is that decoded audio is written directly into a ring buffer and then transferred to the output buffer during render/mix without additional intermediate copies.

From an energy perspective, energy-aware audio design enables the engine to scale quality dynamically based on processor load and battery conditions. Defining Low/Medium/High modes and using triggers such as CPU load, underrun rate, device temperature, and battery state supports dynamic decision-making. Practically, the engine can disable reverb, reduce oversampling, simplify spatialization, or even lower the voice cap to prevent thermal throttling and rapid battery drain.

Conditional processing is also highly effective. If the output is silent or near-silent, parts of the DSP chain can be bypassed; if no voices are active, the engine can enter a sleep state to avoid wasting compute cycles and energy.

To ensure these decisions are genuinely "intelligent," a lightweight feedback loop should run in the non-real-time domain to monitor indicators such as callback time (mean and 95th percentile), underrun count, CPU usage, latency estimates, and active voice count. Based on these signals, the system can adaptively apply actions such as changing the quality mode, bounded buffer-size adjustment, decreasing or increasing the voice cap, and enabling or disabling effects.

3. Findings and Results

Study Model Design

To empirically assess the proposed architecture, a case study with characteristics closely aligned with real-world applications was designed to systematically evaluate the behavior of the real-time audio simulation engine under dynamic and high-load conditions. The selected scenario is an interactive audio scene similar to a game or augmented reality (AR) environment, in which the user generates frequent audio events by tapping the screen or through collisions among virtual objects. Each event triggers a short sound effect (SFX) with variable parameters, such that loudness is a function of collision speed or impact intensity, pitch depends on distance or object type, and panning depends on the horizontal position of the audio source. Concurrently, a background ambience layer and a lightweight looped music track are played to approximate a realistic application workload. The number of events is increased in a controlled manner to evaluate the engine across multiple load levels. Accordingly, three test profiles were defined: in the low-load condition, with 4–8 simultaneous voices and minimal DSP, the engine is expected to operate with negligible latency and temporal variability; in the medium-load condition, with 16–24 simultaneous voices and active filtering and panning,

stability and resource consumption are examined more rigorously; and in the high-load condition, with more than 32 simultaneous voices and lightweight reverb and simplified spatialization enabled, adaptive mechanisms such as voice stealing and quality scaling are expected to engage. For precise evaluation, a set of real-time metrics—including callback execution time (mean and 95th percentile), underrun count, and jitter—were recorded as primary indicators of real-time success or failure. In parallel, resource metrics such as average and peak CPU utilization, memory footprint attributable to pools and buffers, and—where feasible—energy indicators or battery discharge rate were monitored. The perceptual quality dimension was also considered through listening checks to identify clicks, dropouts, or distortion, and to evaluate whether quality reductions during scaling occurred gradually. Finally, to ensure the case study was not merely demonstrative, a pressure-response control policy was defined. As callback time approached a risk threshold or an underrun occurred, the system first reduced effect quality or disabled reverb, then lowered the voice cap, activated priority- and amplitude-based voice stealing, and—if pressure persisted—applied a bounded increase to framesPerBuffer within allowable limits. This policy illustrates how the engine establishes a deliberate and managed trade-off between latency and stability.

Applied Scenario

In this scenario, the user generates multiple audio events via screen taps or collisions among virtual objects. Each event triggers a short sound effect (SFX) with dynamic parameters, such that loudness depends on collision speed, pitch depends on distance or object type, and panning depends on the horizontal position of the audio source. Simultaneously, a background ambience layer and a lightweight looped music track are played to establish a stable workload consistent with realistic conditions. Event frequency is controllable to examine engine behavior under different levels of pressure. Three load profiles were defined: in the low-load condition, with 4–8 simultaneous voices and minimal DSP, the engine is expected to exhibit minimal latency and temporal variability; in the medium-load condition, with 16–24 simultaneous voices and filtering and panning enabled, stability and resource usage are evaluated more stringently; and in the high-load condition, with more than 32 simultaneous voices and lightweight reverb and simplified spatialization enabled, adaptive mechanisms such as voice stealing and quality scaling are expected to activate. The evaluation is based on three classes of metrics: real-time metrics (callback execution time—mean and 95th percentile—underrun count, and jitter), resource metrics (CPU consumption, memory footprint, and energy indicators), and perceptual metrics assessing listening quality, the presence of clicks or dropouts, and the manner in which quality degrades during scaling. To prevent abrupt real-time failure, a pressure-response policy is triggered when callback time approaches a risk threshold or when an underrun occurs. This policy sequentially includes reducing or bypassing reverb, lowering the voice cap, activating voice stealing based on priority and signal amplitude, and—if pressure persists—applying a bounded increase in buffer size. This control logic demonstrates that the engine manages the latency–stability trade-off deliberately rather than incidentally.

Implementation

Android was selected as the primary implementation platform because its hardware diversity and well-known latency and jitter challenges make it an appropriate environment for evaluating real-time architectures. The engine's audio output was implemented using AAudio, with a compatibility path to OpenSL ES enabled on devices where AAudio is unavailable. A target sampling rate of 48 kHz was adopted, with planned compatibility for 44.1 kHz. The real-time core and DSP components were developed in C/C++ to ensure strict control over execution time and memory behavior, while the application layer was implemented in Kotlin/Java, using a build system based on CMake and the Android NDK.

The real-time core operates within a callback loop that performs only time-sensitive operations. At the start of each callback, control messages are consumed in batches from a real-time-safe queue. Next, each voice state—including activation status, envelope, and spatial parameters—is updated; samples are generated or read; multi-source mixing is performed; and a minimal DSP chain is applied. Finally, the output data are written into the output buffer and delivered to the backend. No dynamic memory allocation, I/O operations, or heavy locks are used in this section to preserve execution-time predictability.

The Control layer executes on a non-real-time thread and is responsible for receiving application events, translating them into audio commands (e.g., Play/Stop), and applying adaptive policies such as quality scaling and voice capping. Audio file loading, decoding, and streaming are handled in the Resource/Streaming layer, and their outputs are placed as ready-to-consume PCM chunks in a ring buffer, ensuring that the Audio Thread functions purely as a data consumer.

To ensure stability, a lightweight monitoring loop in the non-real-time domain tracks indicators such as callback time, underrun count, and CPU consumption, and applies decisions such as reducing effect quality, adjusting the voice cap, or making bounded buffer-size changes. This implemented approach indicates that the proposed architecture can maintain a managed balance among latency, stability, and resource consumption in real executions.

Evaluation and Experimental Results

Experiments were conducted on multiple Android devices with different hardware tiers, including mid-range and high-end devices, to assess engine behavior across heterogeneous conditions. The system sampling rate was set to 48 kHz, and the initial buffer size was configured at 256 frames. Each experiment ran for at least 120 seconds to enable evaluation of temporal stability over extended intervals. For a fair comparison, two reference configurations were defined: (1) a baseline implementation consisting of a simple audio engine with no adaptive policies, fixed DSP, and no voice stealing mechanisms; and (2) the proposed implementation incorporating all architectural components described above, including adaptive buffering, voice capping and stealing, virtualization, and quality scaling.

The evaluation used four classes of metrics. Timing metrics included end-to-end latency, audio callback execution time (mean and 95th percentile), and jitter as the measure of temporal variability. Stability metrics included underrun counts and perceptually detectable dropout events. Resource metrics included average and peak CPU usage, memory footprint, and energy indicators or battery discharge rate (where supported by the operating system). Finally, perceptual metrics were assessed via listening evaluations to detect clicks, noise, rhythmic instability, and quality degradation under load.

The timing results indicated that, in the proposed implementation, effective system latency decreased substantially. Under low-load scenarios, latency remained within a range that was effectively imperceptible to users, whereas the baseline implementation exhibited a gradual and noticeable latency increase as the number of simultaneous voices grew. Callback execution-time analysis further showed that the optimized configuration not only reduced mean callback time but also reduced temporal dispersion. In particular, the 95th percentile of callback time remained at a safer margin from the buffer deadline, indicating higher predictability and a lower risk of real-time failure.

In the assessment of real-time stability and load management, the difference between the two approaches became explicit. Under high-load conditions—more than 32 simultaneous voices with effects enabled—the baseline implementation experienced multiple underruns, manifested as perceptible audio dropouts. In contrast, the proposed architecture prevented sustained underruns by activating voice stealing in a timely manner and gradually reducing effect quality. These results indicate that the adaptive approach can manage the trade-off between audio quality and real-time stability in a controlled and predictable manner, rather than allowing abrupt system failure.

4. Discussion and Conclusion

The present study examined the effectiveness of an adaptive, architecture-driven real-time audio simulation engine on mobile platforms with respect to latency, jitter, system stability, resource consumption, and perceptual audio quality. The empirical findings demonstrate that the proposed architecture substantially improves real-time performance compared with a baseline mobile audio engine lacking adaptive mechanisms. Specifically, the optimized engine achieved significantly lower end-to-end latency, reduced temporal variability in callback execution, near-elimination of sustained underruns under high load, and a controlled, perceptually acceptable degradation of audio quality when system pressure increased. These outcomes validate the central premise of this research: that architectural design and adaptive control strategies are decisive factors in achieving reliable and efficient real-time audio processing on constrained mobile hardware.

One of the most prominent findings was the marked reduction in effective latency and jitter in the optimized implementation. The lower mean callback execution time and the narrower dispersion of callback timing—particularly the improved 95th percentile margin relative to buffer deadlines—indicate a substantial increase in execution-time predictability. This aligns with prior research emphasizing that minimizing the critical execution path and isolating time-sensitive workloads are essential for stable mobile multimedia performance (Schobel et al., 2016; Xue & Zheng, 2025). The present results extend this principle specifically to interactive audio systems, demonstrating that architectural separation between real-time and non-real-time domains materially improves timing determinism in practice. These findings are also consistent with broader mobile optimization studies that identify system-level

design as a stronger determinant of real-time behavior than isolated algorithmic improvements (Ota et al., 2017; Zhao et al., 2022).

The observed improvements in real-time stability, particularly under high-load conditions, further reinforce the value of adaptive resource management. Whereas the baseline engine experienced repeated underruns and audible dropouts when the number of concurrent voices exceeded 32, the proposed engine maintained stable output through timely activation of voice stealing, gradual effect quality reduction, and bounded buffer-size adjustments. This behavior reflects the adaptive control philosophy advocated in recent mobile systems research, which stresses dynamic workload management and graceful degradation as necessary responses to resource variability (Deng, 2019; Zhao et al., 2022). In the audio domain, the present findings confirm that such adaptive strategies not only preserve system stability but also prevent abrupt perceptual failures that are highly detrimental to user experience (Collins, 2008; Sweet, 2014).

Importantly, the perceptual evaluation component revealed that users consistently preferred controlled quality reduction over audible instability or dropouts. This result corroborates long-standing observations in game audio and interactive media that continuity and responsiveness dominate user satisfaction more strongly than absolute fidelity (Collins, 2008; Sweet, 2014). It also aligns with contemporary work in spatial and immersive audio, which emphasizes perceptual coherence and temporal consistency as key determinants of presence and engagement (Fırat et al., 2022; Jot et al., 2021; Yang et al., 2022). The present study thus provides empirical support for prioritizing perceptual stability over raw processing complexity in mobile audio engine design.

From a technical perspective, the success of the proposed engine can be attributed largely to its architecture-driven design. The strict isolation of the Audio Thread from non-deterministic operations—such as I/O, dynamic memory allocation, and heavy synchronization—proved instrumental in maintaining predictable execution. This result is consistent with the design principles articulated in real-time audio engineering and computer music literature, which emphasize constant-time operations, minimal branching, and deterministic scheduling as prerequisites for reliable real-time sound processing (Farnell, 2010; Lazzarini et al., 2016; Pulkki & Karjalainen, 2015). However, the present study advances this literature by demonstrating how these principles can be operationalized within the constraints of mobile operating systems and heterogeneous hardware.

The integration of adaptive quality scaling further contributed to system robustness. By selectively degrading computationally intensive effects such as reverb and spatialization under load, the engine preserved timing guarantees without fully sacrificing auditory coherence. This strategy resonates with findings from immersive audio research, which highlight that simplified spatial models and lightweight reverberation often provide sufficient perceptual realism in mobile and AR contexts (Fırat et al., 2022; Yang et al., 2022). It also aligns with emerging trends in mobile deep learning and multimedia optimization, where dynamic model scaling and conditional execution are increasingly employed to balance performance and resource constraints (Latif et al., 2023; Zhao et al., 2022).

Resource consumption metrics further validate the effectiveness of the proposed approach. The controlled reduction in CPU utilization and memory footprint under high-load conditions confirms that adaptive voice management, virtualization, and zero-copy buffering significantly enhance computational efficiency. These results mirror conclusions from mobile multimedia research, which identifies memory bandwidth and CPU cycles as critical bottlenecks on handheld devices (Deng, 2019; Ota et al., 2017). The present findings demonstrate that careful architectural choices at the audio engine level can meaningfully mitigate these constraints without compromising interactive responsiveness.

From a broader system design and management perspective, the implications of these findings are substantial. As interactive audio becomes a central feature of mobile applications—ranging from entertainment and social platforms to AR, training, and collaborative systems—the reliability of real-time audio infrastructure directly influences product success and organizational competitiveness (Khan, 2024; Xue & Zheng, 2025). The demonstrated gains in stability, efficiency, and perceptual quality suggest that investment in robust audio engine architecture yields tangible value at both the technical and strategic levels. This reinforces recent management-oriented perspectives that treat system architecture not merely as an engineering artifact but as a strategic organizational resource (Khan, 2024; Xue & Zheng, 2025).

Furthermore, the study's results complement emerging research on the future of audio ecosystems, including the audio metaverse and interoperable immersive environments. These domains demand scalable, adaptive, and cross-

platform audio infrastructures capable of delivering consistent real-time experiences across diverse devices (Jot et al., 2021; Yang et al., 2022). The proposed architecture, with its emphasis on modular design, adaptive control, and platform-aware optimization, provides a practical foundation for meeting these emerging requirements.

In summary, the empirical evidence supports the conclusion that architecture-driven, adaptive real-time audio engine design significantly enhances performance, stability, and user experience on mobile platforms. The convergence of reduced latency, minimized jitter, improved resource efficiency, and perceptually acceptable quality degradation demonstrates that robust system design can overcome many of the inherent limitations of mobile hardware. These findings contribute to both the technical literature on real-time audio processing and the managerial discourse on system design as a strategic capability.

This study is subject to several limitations. First, although multiple Android devices were tested, the hardware sample cannot fully represent the vast diversity of mobile devices and operating system configurations currently in use. Second, the perceptual evaluation relied on controlled listening assessments rather than large-scale user studies, which may limit the generalizability of the subjective findings. Third, the experiments focused primarily on interactive audio workloads similar to games and AR scenarios; results may differ for other application domains such as teleconferencing or large-scale collaborative systems. Finally, long-term energy consumption effects under prolonged real-world usage were not comprehensively assessed.

Future research should extend this work by conducting large-scale user studies to quantify perceptual outcomes across diverse demographic groups and usage contexts. Comparative evaluations on additional mobile platforms and operating systems would strengthen external validity. Further investigation into the integration of machine learning–based audio processing within adaptive real-time architectures could yield valuable insights, particularly regarding dynamic quality control and predictive resource management. Longitudinal studies examining battery health, thermal behavior, and user retention over extended deployment periods are also recommended.

Practitioners should prioritize architectural separation between real-time and non-real-time domains when designing mobile audio systems. Adaptive control mechanisms should be treated as first-class components rather than optional optimizations. Development teams are encouraged to invest in profiling, monitoring, and automated quality-scaling pipelines early in the product lifecycle. Finally, management should recognize audio infrastructure as a strategic asset that directly influences product quality, user satisfaction, and competitive advantage.

Authors' Contributions

Authors contributed equally to this article.

Declaration

In order to correct and improve the academic writing of our paper, we have used the language model ChatGPT.

Transparency Statement

Data are available for research purposes upon reasonable request to the corresponding author.

Acknowledgments

We would like to express our gratitude to all individuals helped us to do the project.

Declaration of Interest

The authors report no conflict of interest.

Funding

According to the authors, this article has no financial support.

Ethics Considerations

In this research, ethical standards including obtaining informed consent, ensuring privacy and confidentiality were considered.

References

Collins, K. (2008). *Game sound: An introduction to the history, theory, and practice of video game music and sound design*. MIT Press. https://doi.org/10.7551/mitpress/7909.001.0001

Deng, Y. (2019). Deep learning on mobile devices: a review. Mobile Multimedia/Image Processing, Security, and Applications 2019,

Farnell, A. (2010). *Designing sound*. MIT Press. https://books.google.com/books?id=eMPxCwAAQBAJ&source=gbs_navlinks_s

Fırat, H. B., Maffei, L., & Masullo, M. (2022). 3D sound spatialization with game engines: the virtual acoustics performance of a game engine and a middleware for interactive audio design. *Virtual Reality*, *26*(2), 539-558. https://doi.org/10.1007/s10055-021-00589-0

Jahangashteh, E., Ghadri, A., Davari, R., & Jalalvand, M. (2022). A Study of Mobile Operating Systems. The 16th National Conference on Computer Science, Engineering, and Information Technology, Babol.

Jot, J. M., Audfray, R., Hertensteiner, M., & Schmidt, B. (2021). Rendering spatial sound for interoperable experiences in the audio metaverse. 2021 Immersive and 3D Audio: from Architecture to Automotive (I3DA),

Khan, K. (2024). Advancements and Challenges in 360 Augmented Reality Video Streaming: A Comprehensive Review. *International Journal of Computing*, *13*(1), 1-20. https://doi.org/10.30534/ijccn/2024/011312024

Latif, S., Shoukat, M., Shamshad, F., Usama, M., Ren, Y., Cuayáhuitl, H., Wang, W., Zhang, X., Togneri, R., Cambria, E., & Schuller, B. W. (2023). Sparks of large audio models: A survey and outlook. *arXiv preprint*. https://arxiv.org/abs/2308.12792

Lazzarini, V., Timoney, J., & Keller, D. (2016). *Computer music instruments*. Springer. https://doi.org/10.1007/978-3-319-63504-0

Ota, K., Dao, M. S., Mezaris, V., & Natale, F. G. D. (2017). Deep learning for mobile multimedia: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, *13*(3s), 1-22. https://doi.org/10.1145/3092831

Pulkki, V., & Karjalainen, M. (2015). *Communication acoustics*. Wiley. https://doi.org/10.1002/9781119825449

Schobel, J., Pryss, R., Schickler, M., & Reichert, M. (2016). A lightweight process engine for enabling advanced mobile applications. OTM Confederated International Conferences" On the Move to Meaningful Internet Systems",

Sweet, M. (2014). *Writing interactive music for video games*. Addison-Wesley. https://books.google.com/books?id=CQqSBAAAQBAJ&source=gbs_navlinks_s

Xue, M., & Zheng, Y. (2025). Exploring Updating Functional and Design Requirements of Audio Across Diverse Scenarios. International Conference on Human-Computer Interaction,

Yang, J., Barde, A., & Billinghurst, M. (2022). Audio augmented reality: A systematic review of technologies, applications, and future research directions. *Journal of the Audio Engineering Society*, *70*(10), 788-809. https://doi.org/10.17743/jaes.2022.0048

Zhao, T., Xie, Y., Wang, Y., Cheng, J., Guo, X., Hu, B., & Chen, Y. (2022). A survey of deep learning on mobile devices: Applications, optimizations, challenges, and research opportunities. *Proceedings of the IEEE*, *110*(3), 334-354. https://doi.org/10.1109/JPROC.2022.3153408