Katonic

THOUGHT LEADERSHIP

The Hidden Architecture Problem in Voice AI

Why State Ownership, Not Model Quality, Determines Production Success

Anuja Fole | Katonic AI Engineering Team

February 2026

SECTION 01

Executive Summary

The voice AI market is projected to reach $13 billion by 2032, yet enterprise deployments continue to struggle with production reliability. While industry attention focuses on model quality, latency benchmarks, and feature comparisons, the most critical factor in production success is often overlooked: state ownership clarity.

This paper presents findings from Katonic AI's engineering work integrating the Orpheus Arabic-Saudi TTS model into our Katonic Studio. What began as a routine model integration revealed systemic architecture challenges that affect any real-time voice AI deployment.

57%	$100B+	<500ms
Companies with AI agents in production	Projected AI agents market by 2032	Required voice-to-voice latency

We didn't fix audio. We fixed state ownership. Real-time systems don't fail because of missing features. They fail because ownership is unclear.

The lessons learned apply broadly to any enterprise deploying real-time AI agents, voice assistants, or streaming media applications.

SECTION 02

1. The Voice AI Production Gap

Enterprise voice AI adoption is accelerating rapidly. According to recent industry research, 57% of companies already have AI agents running in production, and AI agents are projected to grow from $3.6 billion in 2024 to over $100 billion by 2030.

Yet a significant gap exists between proof-of-concept and production deployment:

- Only 11% of enterprises have achieved full agentic AI deployment despite 65% having pilots
- Integration complexity remains the top barrier cited by enterprise leaders
- The performance bar is exceptionally high: sub-500ms voice-to-voice latency required for natural interaction
- Production requirements (scaling, governance, reliability) far exceed demo requirements

The Real Challenge

Industry analysts correctly note that cheap, fast, high-quality voices alone don’t automatically translate into great real-time conversational products. A production-grade agent must capture audio, transcribe it, plan a response, stream synthetic speech back, and handle interruptions—all in real time.

But even this framing understates the challenge. The real issue isn’t the complexity of the pipeline. It’s the clarity of ownership at every stage.

SECTION 03

2. Case Study: Integrating Orpheus Arabic-Saudi TTS via Groq

Katonic Studio enables enterprises to integrate any model, including specialized TTS models beyond standard OpenAI voice options. When integrating the canopylabs/orpheus-arabic-saudi model via Groq's inference infrastructure, we encountered challenges that illuminate broader patterns in real-time system design.

Groq's API delivers Orpheus TTS with exceptional speed, achieving up to 100 characters per second with sub-200ms time-to-first-byte latency. The model offers four distinct Saudi dialect voices (Fahad, Sultan, Lulwa, and Noura) with authentic pronunciation and regional nuances. On paper, this combination of Groq's inference speed and Orpheus's voice quality should deliver an exceptional user experience.

Why Groq + Orpheus
Groq's LPU (Language Processing Unit) inference engine delivers industry-leading speed for AI workloads. Combined with Canopy Labs' Orpheus Arabic-Saudi model, enterprises can offer authentic Saudi dialect TTS with emotional expressiveness and low latency. This integration represents the cutting edge of multilingual voice AI.

The Initial Implementation

The original system appeared straightforward: text sent to Groq's speech endpoint, Orpheus TTS processing the request, audio chunks streaming back over WebSocket, and the browser playing them. Groq's infrastructure handled the model inference brilliantly, delivering audio chunks with minimal latency.

In development, it worked perfectly. In production, problems emerged:

Symptom	Frequency in Production
First message plays, second message silent	~30% of sessions
Audio streams never completing	Intermittent
UI stuck in 'speaking' state indefinitely	~15% of sessions
STOP button not reliably stopping playback	Frequent
WebSocket connections accumulating	Memory leak over time

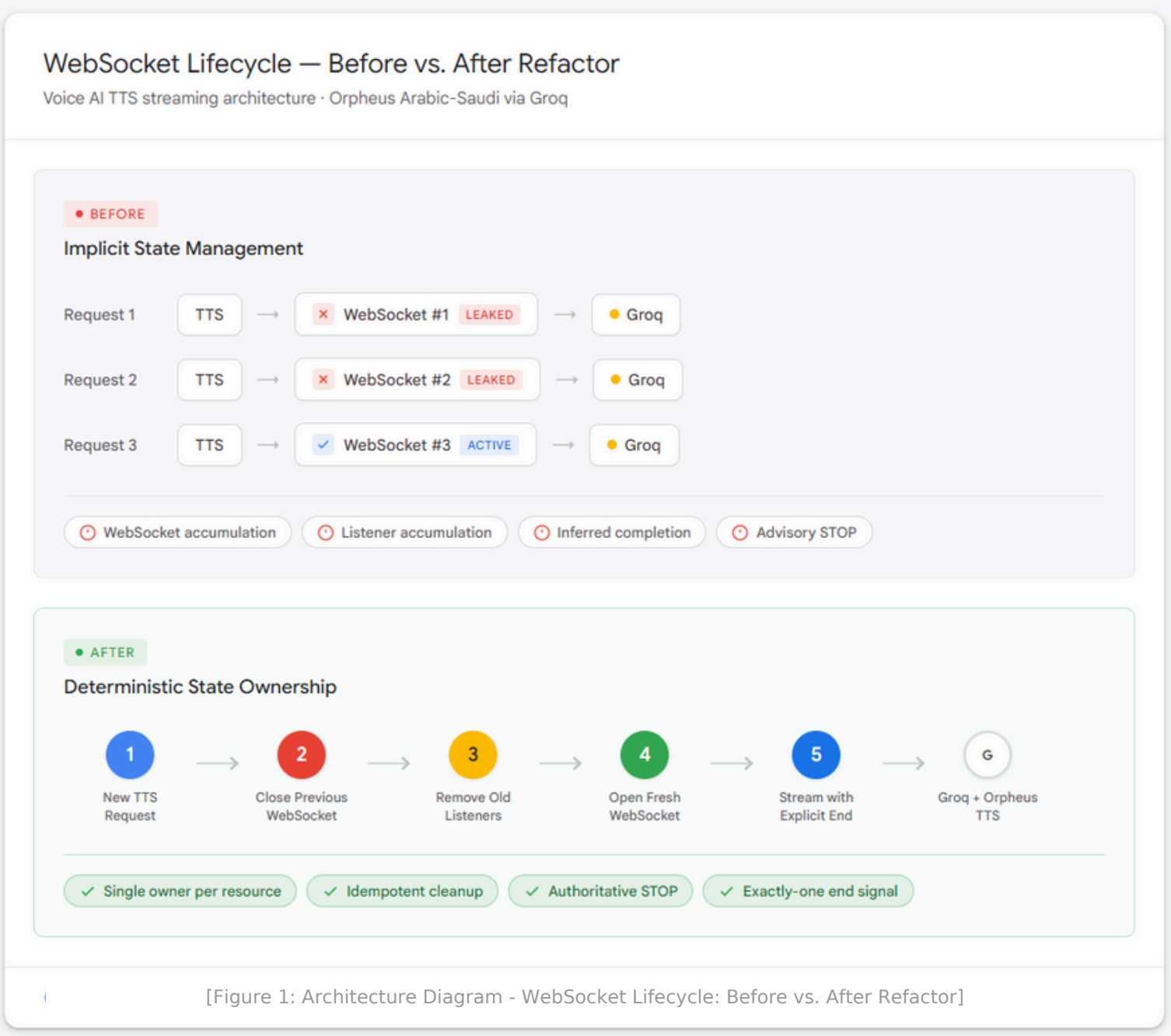
Root Cause Analysis

These weren't five separate bugs. They were symptoms of one architectural problem: unclear state ownership.

Backend issues included: every TTS request creating a new WebSocket without closing previous ones; event listeners accumulating because old ones were never removed; stream completion being inferred

rather than explicitly signaled; and STOP commands being treated as advisory rather than authoritative.

Frontend issues included: complex buffering logic attempting to be ‘smart’ about audio chunks; multiple async code paths competing for control; playback timing based on heuristics rather than deterministic scheduling; and UI state depending on assumptions rather than explicit signals.



[Figure 1: Architecture Diagram - WebSocket Lifecycle: Before vs. After Refactor]

SECTION 04

3. The Solution: Deterministic State Ownership

The refactor centered on one principle: make lifecycles explicit and deterministic.

Backend Architecture Changes

Single WebSocket Per Session

Instead of creating new WebSocket connections blindly, we store the connection reference on the socket itself. When a new TTS request arrives, the previous connection is explicitly closed, all related listeners are removed, and a fresh session starts cleanly.

> **Key Principle**
> Every resource (WebSocket, audio context, stream) must have a single, explicit owner. When ownership transfers, the previous owner must explicitly release the resource.

Centralized, Idempotent Cleanup

We introduced a single cleanup function that clears pending timeouts, sends a final stream-end signal, closes the external WebSocket safely, and can be called multiple times without side effects. This single function replaced scattered cleanup logic spread across multiple event handlers.

Authoritative STOP

Previously, STOP was advisory. Now it’s authoritative. When the client sends STOP, the external WebSocket is immediately closed, the audio stream is terminated, and the client is notified of stream completion—all in a deterministic sequence.

Exactly One Stream End Signal

Regardless of how the external service finishes (text markers, binary silence, manual STOP, client disconnect), the frontend receives exactly one end-of-stream signal. This contract proved to be the most important architectural decision in the entire refactor.

Frontend Architecture Changes

PCM-Native, Timeline-Driven Playback

We abandoned the complex buffering approach entirely. The new implementation accepts raw PCM only, converts Int16 to Float32, and schedules each chunk directly on the AudioContext timeline at precise offsets. This eliminated buffering bugs and timing issues.

Explicit Stream Completion

Playback completes only when the end-of-stream signal is received AND all scheduled audio sources finish playing. As a safety net, we added a timeout in case the browser’s onended event doesn’t fire. This provides a deterministic guarantee: the caller knows exactly when the voice has finished speaking.

Message-Scoped Audio Sessions

Every new message resets the audio timeline completely, clears previous sources, and starts fresh. This permanently fixed the ‘second message is silent’ bug.

SECTION 05

4. Broader Implications for Enterprise Voice AI

The Architecture vs. Model Debate

Industry discussion often focuses on model selection: which TTS model has the lowest latency, the most natural voice, the best multilingual support. These factors matter. Groq's inference infrastructure genuinely enables new categories of real-time applications. Orpheus delivers authentic Arabic speech that wasn't previously available at production speeds.

As one industry analysis noted, the main competitive frontier is infrastructure: who can deliver voices at scale with the lowest latency and least friction. We would extend this: the competitive frontier is also architectural. Infrastructure without clear state management creates fast failures instead of slow ones.

The Three Laws of Real-Time State Management

Based on this work and broader patterns in real-time systems, we propose three principles:

- **Ownership Law:** Every resource must have exactly one owner at any given time. When ownership transfers, the previous owner must explicitly release the resource.
- **Cleanup Law:** All cleanup functions must be idempotent. It must be safe to call cleanup multiple times without side effects.
- **Completion Law:** Stream completion must be explicit, never inferred. The system must guarantee exactly-once delivery of completion signals.

These laws apply beyond voice AI to any real-time streaming system: video conferencing, collaborative editing, live dashboards, multiplayer gaming, or IoT telemetry.

Why Demos Work and Production Fails

Demos work because: network conditions are ideal; users follow expected patterns; sessions are short; only one person is testing; and there's no accumulated state from previous sessions.

Production fails because: network jitter introduces timing variations; users interrupt, retry, and behave unpredictably; sessions last longer and span multiple interactions; concurrent users create resource contention; and state accumulates across hundreds of sessions.

The gap between demo and production is the gap between implicit assumptions and explicit contracts.

SECTION 06

5. Implementation Guidance for Enterprise Teams

Checklist for Real-Time Voice AI Deployments

Requirement	Verification Method
Single WebSocket per user session	Connection count monitoring
Explicit cleanup on new session start	Log cleanup calls
Idempotent cleanup functions	Call cleanup 3x in tests
Exactly-once stream end signal	Count end signals in tests
STOP command is authoritative	Verify immediate termination
Timeline-driven audio scheduling	No overlap in audio output
Message-scoped audio sessions	Test rapid consecutive messages
Safety timeout for completion	Simulate browser event failures

Testing Recommendations

Standard unit and integration tests are insufficient for real-time systems. We recommend:

- Multi-session stress tests: Send 100 consecutive messages and verify no resource leaks
- Interrupt tests: Issue STOP commands at random points during streaming
- Network chaos tests: Introduce latency and packet loss to verify graceful degradation
- Long-running soak tests: Run continuous sessions for hours and monitor resource consumption

SECTION 07

6. Conclusion

The voice AI industry is at an inflection point. Models are becoming commoditized. The TTS market offers dozens of options with sub-200ms latency and near-human quality. Infrastructure providers like Groq are making model serving essentially a solved problem.

Our experience integrating Orpheus Arabic-Saudi TTS via Groq reinforced a lesson that applies far beyond voice: real-time systems succeed or fail based on state ownership clarity. Groq delivered exceptional inference speed. Orpheus delivered authentic Arabic voices. But the system didn't work reliably until we fixed how state was owned, transferred, and released.

For enterprises evaluating voice AI platforms, we suggest looking beyond model benchmarks to ask:

- How does the platform handle session lifecycle management?
- What happens when users interrupt mid-stream?
- How are stream completion signals guaranteed?
- What testing methodology verifies production reliability?

The answers to these questions will predict production success far more reliably than any latency benchmark.

If you can't clearly explain when something starts and when it ends, it will break in production.

SECTION 08

About Katonic AI

Katonic AI is the sovereign enterprise AI platform, providing the operating system for full-stack AI agents. Our platform enables enterprises to build, deploy, and govern AI agents with complete freedom: any framework, any model, any cloud. Deployed 100% on your infrastructure with zero data egress, Katonic serves regulated industries across 11 countries.

Katonic Studio allows enterprises to integrate any model, including specialized TTS models like Orpheus Arabic-Saudi, into production-ready applications with enterprise-grade reliability, security, and governance.

Key Capabilities

- 250+ models supported with instant swapping
- 80+ pre-built AI agents with full source code
- 100+ MCP servers for enterprise tool connectivity
- 50+ knowledge connectors for enterprise data
- On-premise, private cloud, hybrid, and air-gapped deployment options
- ISO 27001 certified, FIPS 140-2 compliant, HIPAA, GDPR, EU AI Act

Learn more at www.katonic.ai

SECTION 09

References and Further Reading

Industry Reports and Market Analysis:

- TTS AI Model Market Outlook 2025–2032, Intel Market Research
- State of Voice AI 2024, Cartesia
- The State of AI Agents in Enterprise Q3 2025, Lyzr
- G2’s Enterprise AI Agents Report 2026
- Voice AI Market Map, Sierra Ventures

Technical References:

- WebSocket Architecture Best Practices, Ably
- TTS Voice AI Model Guide 2025, Layercode
- Orpheus TTS Documentation, Canopy Labs
- Groq Text-to-Speech API Documentation
- OpenAI Realtime API Documentation

The Enterprise Operating System

for Full-Stack AI Agents

www.katonic.ai