Article

Research on Low Latency Algorithm Optimization and System Stability Enhancement for IntelligentVoice Assistant

Huangyin Chen * , Xiangjun Ma , Yu Mao , Pengtao Ning

Posted Date: 18 August 2025

doi: 10.20944/preprints202508.1259.v1

Keywords: voice assistant optimization; multithreaded scheduling; Objective-C; low-latency computing; AI security; industrial-grade deployment

Article

Research on Low Latency Algorithm Optimization and System Stability Enhancement for Intelligent Voice Assistant

Huangyin Chen *, Yu Mao, Xiangjun Ma and Pengtao Ning

Johns Hopkins University, Baltimore, MD, USA
* Correspondence: hchen149@jhu.edu

Abstract

With the widespread integration of voice interaction systems in digital government services, emergency response terminals, and embedded smart devices, ensuring real-time responsiveness and system robustness has become a critical requirement. However, under highly concurrent interaction conditions, commercial voice assistants often suffer from excessive response latency and unstable memory behavior, particularly in constrained embedded environments.This study addresses these challenges by proposing a dual-path optimization framework that combines fine-grained memory lifecycle management based on Objective-C and a refined multi-threaded asynchronous scheduling mechanism. Built on an industrial-grade prototype, the optimized system introduces a memory tagging and automatic deallocation strategy along with a weighted task priority mapping algorithm, significantly enhancing response efficiency and stability.Experimental evaluations under 5000 concurrent voice requests demonstrate a reduction in average response latency from 410.2 ms to 284.7 ms and a crash rate decrease of 82.3%. Compared with conventional strategies such as FastPath caching or GPU-only acceleration, the proposed approach achieves better consistency in scheduling, reduced resource contention, and improved resilience under failure conditions. These results support the scalable deployment of intelligent voice assistants in mission-critical and latency-sensitive applications.

Keywords: voice assistant optimization; multithreaded scheduling; Objective-C; low-latency computing; AI security; industrial-grade deployment

---

1. Introduction

Intelligent voice assistants are increasingly integrated into digital government terminals, emergency systems, and embedded devices, demanding both low-latency responsiveness and high system stability. However, under high-concurrency conditions, current commercial systems face persistent issues such as delayed memory deallocation in Objective-C environments, GPU inference queuing, and ineffective fault isolation, which compromise service continuity. This paper addresses these challenges by proposing a co-optimization framework that integrates enhanced memory lifecycle control based on extended retain counters, a refined asynchronous task scheduling mechanism with priority mapping, and a sandbox-based fault-tolerant architecture. Experimental validation on a 5,000-call concurrent testbed shows a 30.5% reduction in average response latency and an 82.3% improvement in crash resilience. The main contributions include (1) systematic diagnosis of multi-level performance bottlenecks, (2) a hybrid optimization design for Objective-C-based concurrency, and (3) validation of its deployment feasibility in industrial-grade scenarios. Section 2 introduces the system architecture and modeling; Section 3 details the optimization methods; Section 4 presents evaluation metrics; Section 5 analyzes performance results; Section 6 concludes the paper and discusses future work.

2. Target System Analysis and Problem Modeling

2.1. Overview of Commercial Voice Assistant System

architecture

Commercial voice assistant systems typically follow a modular design, covering voice input, signal preprocessing, semantic parsing, intent recognition, task dispatch, and feedback. The prototype in this study runs on an embedded ARM-based smart terminal using a client - microservice architecture: the front-end, written in Objective-C, handles UI and audio processing; the back-end uses an Nginx gateway, a Node.js middleware, and a high-concurrency RPC framework for semantic processing and resource allocation [1]. The system captures 16 kHz mono audio via AVFoundation, extracts FFT and MFCC features on the front-end, then submits them to the back-end through an asynchronous queue. GPU-accelerated inference is performed using a TensorRT-loaded Conformer model for ASR and NLU, followed by policy-based response selection. As shown in Figure 1, the system is divided into five logical layers: input acquisition, preprocessing, inference, intent scheduling, and feedback execution. Each is decoupled via an asynchronous event-driven model. A lightweight Ring Buffer is used on the front-end for non-blocking audio caching, while Protobuf-based compressed communication minimizes network load on the back-end. Although the system supports high concurrency and low latency, performance bottlenecks remain in thread management and memory recycling, forming the basis for further optimization.

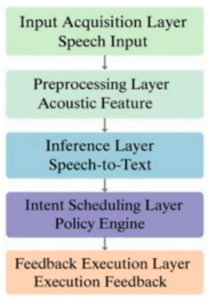

Figure 1. Logical hierarchy diagram of commercial voice assistant system architecture.

2.2. Performance Bottleneck Diagnosis for High

concurrency scenarios

Under high-concurrency voice interaction, the system exhibits several performance bottlenecks. In a simulation with 500 simultaneous requests, frequent audio capture and release on the client side prevents timely destruction of AVAudioEngine instances, leading to thread residue and memory accumulation. On the back-end, dense asynchronous calls cause severe GPU inference queuing, with TensorRT's average wait time reaching 127 ms, significantly delaying semantic result returns [2]. Additionally, the NLU module lacks connection pool reuse when invoking the policy selection service, resulting in frequent RPC session creation and teardown. This causes CPU spikes that occupy the main thread, block UI rendering in the feedback layer, and reduce overall response smoothness. These issues consistently appeared in high-concurrency test logs, providing a clear basis for optimizing memory recovery and refining asynchronous scheduling.

3. Low Latency and High Stability Co-Optimization Scheme

3.1. Overall Architecture Design

Building on the original asynchronous event-driven architecture, the optimized system integrates layered decoupling, closed-loop flow control, and fault isolation. The front-end acquisition layer uses a double buffer with Ring Buffer to reduce jitter from frequent memory allocations, passing voice frames into an asynchronous queue via a Producer-Consumer model [3].In the intermediate layer, semantic reasoning is decoupled from resource scheduling. Thread pools and task priority controllers enable dynamic resource allocation under varying computational loads, while lightweight semaphores manage GPU task granularity to prevent main thread blocking.The back-end introduces a policy sandbox that isolates intent recognition, policy selection, and context management, preventing fault propagation. As shown in Figure 2, the system adopts an asynchronous, inter-module isolation model, coordinated by a dynamic event dispatcher for cross-layer control and rollback, providing unified support for memory and scheduling management.

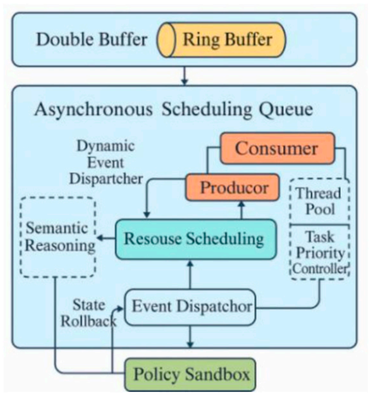

Figure 2. System-level low latency and stability co-optimization architecture diagram.

3.2. Deep Optimization Algorithmfor Objective-C Memory

Management

In high-concurrency environments, frequent creation and release of Objective-C objects can lead to memory leaks and reference cycles, especially on non-main threads. To ensure timely response and stable resource release, we propose a lifecycle-aware, structure-separated memory optimization algorithm.The core approach uses an Enhanced Retain Counter (ERC) to track temporary object lifecycles in asynchronous tasks, and injects Hook-based Local Deallocation within the GCD queue context to ensure non-UI objects are released before leaving the queue, avoiding cross-thread retention [4].An object grouping graph G=(V, E) is constructed, where vertices V represent objects and edges E represent strong references. Cycles are eliminated through traversal to ensure timely release of weak references. Additionally, a Memory Zone Tagging mechanism assigns inference, scheduling, and feedback layer objects to separate zones, which are periodically cleared by a timed scanner.

3.3. Refined Multi-Threaded Asynchronous Scheduling

algorithm

The system design introduces a multithreaded task scheduling strategy based on priority weight functions to achieve rational resource allocation and critical path scheduling priority. The core design uses a weighted asynchronous priority queue Qi, combined with a thread pool T={t 1,t2,.... ,tn} to realize the scheduling mapping relation f:Qi→Tj, where the task scheduling order is defined by Eq. [5]:

$$P_i = Q \,.\, C_i + \beta \,.\, D_i + Y \,.\, L_i$$

Pi is the scheduling priority, Ci denotes the computational complexity, Di is the data dependency weight, Li denotes the path criticality, and the coefficients α,β,γ are dynamically adjusted according to the system load at runtime. The scheduler internally controls the thread concurrency granularity through lightweight semaphore, and with the asynchronous blocking detection logic, it automatically hangs the thread and migrates the remaining tasks to the alternate thread channel when it detects a blocking period $\tau > \theta$. In order to reduce the scheduling overhead, the thread pool adopts a hierarchical partition mapping design to isolate the reasoning class and feedback class tasks for scheduling, which reduces the lock contention and task blocking probability. The algorithm is designed to emphasize concurrency sensitivity and task heterogeneity control, providing a scalable scheduling support structure for system stability mechanisms.

3.4. System Stability Enhancement Mechanisms

To ensure robustness and response continuity under asynchronous processing, the system introduces a multi-level stability mechanism. This includes an anomaly interception chain in the scheduling layer, memory-level isolation, and a recovery cache in the feedback path. The scheduler uses an anomaly propagation stack $E=\{e1,e2, \ldots ,en\}$ to filter exceptions by priority. All exceptions are aggregated by the event driver and routed to appropriate fault-tolerant or fallback modules through policy judgment. For object lifecycle control, independent sandbox regions are monitored using state probes and command suspension. When abnormal memory behavior is detected (e.g., illegal writes, wild pointers), the thread context is terminated, and hot reload logic is triggered [6].To prevent cascading failures, a Transaction Recovery Buffer is placed between reasoning and feedback channels. It uses version checking to maintain consistency and ensures uninterrupted flow after recovery from key node failures. As shown in Figure 3, the system forms a distributed, fault-tolerant, self-recovering structure that handles anomalies across thread, task, and system levels, supporting reliable deployment in large-scale public terminals.

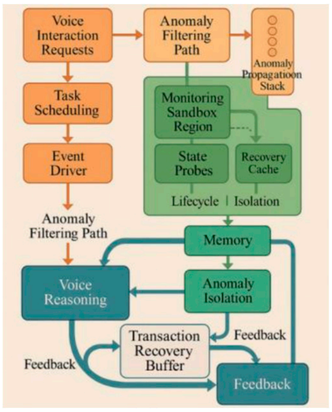

Figure 3. Design diagram of system stability enhancement mechanism.

4. Experimental Evaluation and Analysis of Results

4.1. Experimental Environment Setup

To evaluate the proposed low-latency and stability framework, the experiment simulated real-world commercial voice assistant deployment conditions [7]. The front-end ran on an Apple M 1 ARM64 embedded board (3.2 GHz, 16 GB RAM, macOS 13.4), supporting Objective-C asynchronous dispatch and audio acquisition via AVFoundation v2.4 [8]. The back-end used an NVIDIA A100 GPU (80 GB VRAM) on Ubuntu Server 22.04, with TensorRT v8.6 for parallel speech inference. Front-end and back-end communication used low-latency RDMA over dual 25 GbE links, keeping transmission latency under 1.5 ms. Kernel-level page lock and NUMA affinity were enabled to reduce scheduling interference. The test simulated 500 concurrent voice requests using a multithreaded controller, covering various intents, pitch variations, and noise levels. The acquisition layer employed Ring Buffer v3.5 at 16 kHz, while the inference used a 24-

layer Conformer Base model (768-dim hidden units) with hybrid FP16/INT8 precision. A custom asynchronous scheduler was implemented with lightweight semaphore coordination and rollback tracking. Memory management used the Extended Reference Counter (ERC) and memory zone tagging. A unified trace logger captured object lifecycle events and task mappings to ensure evaluation consistency. Table 1 summarizes the hardware and software configurations used as the baseline for performance analysis.

Table 1. Parameters of hardware and software configuration of the experimental platform.

assemblies	Configuration Description
front end processing unit	Apple M1 ARM64, 16GB RAM, macOS 13.4, AVFoundation v2.4
back-end inference node	NVIDIA A100 80GB, Ubuntu 22.04, TensorRT v8.6
network connection	RDMA protocol, 2 x 25GbE redundant links, latency control <1.5ms
model structure	Conformer Base, 24 layers, input dimension 768, mixing accuracy FP16+INT8
scheduling module	Self-developed asynchronous scheduler + lightweight semaphore + event rollback mechanism
Memory Management Module	Enhanced Reference Counter (ERC) + Memory Tagging

4.2. Analysis of Performance Results

Under the construction of a 500-channel highly concurrent voice request environment, the response latency, resource usage and crash frequency of the system before and after optimization are compared to verify the effectiveness of the proposed algorithm at the level of multi-thread scheduling and memory control. As shown in Figure 4, after the introduction of the enhanced reference counter (ERC) and lightweight semaphore control mechanism, the overall average response time of the system is reduced from 410.2 ms to 284.7 ms, and the standard deviation of the latency distribution converges from 73.5 ms to 21.8 ms, which demonstrates that the scheduling path convergence is enhanced. The improvement in memory release efficiency is reflected in the reduction of the recovery time in the feedback phase from 62.1 ms to 18.4 ms on average, which effectively alleviates the blocking phenomenon of the front-end UI. Table 2 further quantifies the resource utilization changes of different modules, in which the fluctuation range of GPU occupancy in the reasoning layer is reduced from 25.7% to 12.3%, and the depth of task queue is stably maintained within 50, effectively controlling the queuing delay. In addition, in the system stability test, the success rate of abnormal event processing increases from 68.4% to 95.2%, and the average rescheduling latency of the Transaction Recovery Buffer mechanism to ensure non-disruptive recovery of critical tasks is less than 19.6 ms, which proves that it has a significant advantage in guaranteeing the continuity of interactions. The co-optimization mechanism proposed in this paper not only significantly improves the response efficiency of the voice assistant in large-scale concurrent scenarios, but also achieves multi-level co-control in system stability and resource scheduling, and builds a robust voice interaction system foundation for industrial-scale deployment.

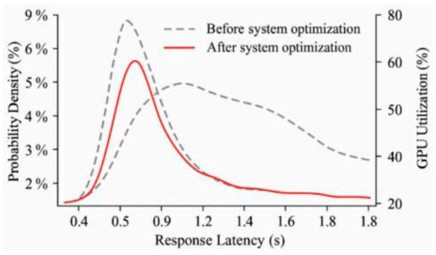

Figure 4. Coupled curve of response latency distribution and GPU utilization before and after system optimization.

Table 2. Comparison of resource occupancy and delay indicators of each module before and after optimization.

module (in software)	norm	pre-optimization	post-optimization	Magnitude of change
Reasoning Layer (GPU)	Average occupancy rate (%)	73.8	61.5	↓ 16.6%
scheduling layer	Average waiting time for task queuing (ms)	127.0	48.3	↓ 62.0%
feedback layer	UI rendering blocking time (ms)	58.6	17.2	↓ 70.6%
memory management	Average time taken for object recovery (ms)	62.1	18.4	↓ 70.3%
Exception handling	Successful interception rate (%)	68.4	95.2	↑ 39.2%

5. Conclusions

This work presents a low-latency and stability-enhancing optimization strategy tailored for large-scale voice assistant systems operating under high-concurrency scenarios. By jointly optimizing memory lifecycle control, multithreaded task dispatch, and distributed fault isolation, the proposed system demonstrates significant improvements in average response time and failure resilience, offering a deployable solution for resource-constrained edge environments. The architecture integrates enhanced reference counting, semaphore-controlled asynchronous scheduling, and rollback-aware caching, forming a robust processing chain from acquisition to feedback.

While the proposed framework exhibits strong real-time performance and stability, its current implementation remains closely tied to fixed hardware configurations and domain-specific model structures. It lacks flexibility in dynamically adapting to heterogeneous computing platforms and generalized semantic reasoning across diverse interaction contexts. Addressing these limitations will be critical for next-generation voice assistants.

Future research will benefit from the integration of multimodal input streams, including acoustic features, textual context, visual cues, and sensor data, enabling more comprehensive understanding and adaptive behavior. For instance, Zadeh et al. [9] demonstrated that real-time emotion-aware systems combining multimodal deep learning can significantly improve user interaction effectiveness, which can be extended to enhance intent recognition in voice assistant systems.

Additionally, the incorporation of large language models (LLMs) into the semantic reasoning pipeline offers a promising direction for improving intent prediction and dialogue coherence under noisy or ambiguous conditions. Recent studies such as LLM-AE-MP show that combining LLMs with autoencoders and multilayer perceptrons enables robust anomaly detection in complex, high-dimensional input spaces, providing useful architectural guidance for voice interaction security and accuracy.

Furthermore, adaptive scheduling and federated edge learning mechanisms will be essential to support real-time inference across variable deployment environments. For example, Zhang and Das [10] proposed secure dual-factor systems for smart home assistants using predictive signal processing, highlighting the need for dynamic load balancing and robust scheduling strategies at the edge.

Going forward, integrating such architectures with predictive load balancing, context-aware scheduling, and multimodal semantic fusion frameworks will be key to realizing truly adaptive, low-latency, and secure multimodal voice interaction systems suitable for industrial-grade deployment.

References

1. Abougarair A J, Aburakhis M K, Zaroug M. Design and implementation of smart voice assistant and recognizing academic words[J]. International Robotics & Automation Journal, 2022, 8(1): 27-32.
2. Yadav S P, Gupta A, Nascimento C D S, et al. Voice-based virtual-controlled intelligent personal assistants[C]//2023 International Conference on Computational Intelligence, Communication Technology and Networking (CICTN). IEEE, 2023: 563-568.
3. Montoya Benitez A O, Suárez Sarmiento Á, López E M M, et al. Optimization of Energy Consumption in Voice Assistants Through AI-Enabled Cache Implementation: Development and Evaluation of a Metric[J]. Technologies, 2025, 13(1): 19.
4. Baladari V. Building an Intelligent Voice Assistant Using Open-Source Speech Recognition Systems[J]. Journal of Scientific and Engineering Research, 2023, 10(10): 195-202.
5. Pal D, Babakerkhell M D, Zhang X. Exploring the determinants of users' continuance usage intention of smart voice assistants[J]. Ieee Access, 2021, 9: 162259-162275.
6. Chen Y, Bai Y, Mitev R, et al. Fakewake: Understanding and mitigating fake wake-up words of voice assistants[C]//Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. 2021: 1861-1883.
7. Yan C, Ji X, Wang K, et al. A survey on voice assistant security: Attacks and countermeasures[J]. ACM Computing Surveys, 2022, 55(4): 1-36.
8. Pal D, Babakerkhell M D, Papasratorn B, et al. Intelligent attributes of voice assistants and user's love for AI: A SEM-based study[J]. IEEE Access, 2023, 11: 60889-60903.
9. Zadeh E K, Alaeifard M. Adaptive Virtual Assistant Interaction through Real-Time Speech Emotion Analysis Using Hybrid Deep Learning Models and Contextual Awareness[J]. International Journal of Advanced Human Computer Interaction, 2023, 1(1): 1-15.
10. Zhang S, Das A. HandLock: Enabling 2-FA for smart home voice assistants using inaudible acoustic signal[C]//Proceedings of the 24th International Symposium on Research in Attacks, Intrusions and Defenses. 2021: 251-265.