© MAR 2026 | IRE Journals | Volume 9 Issue 9 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I9-1715544

AI Voice Call Assistant for Automated Call Management

ASIQ SIKKANDER T N[1], D. REVATHY[2]
[1]PG Student, Department of Computer Applications, SRM Valliammai Engineering College, Chennai
[2]Assistant Professor, Department of Computer Applications, SRM Valliammai Engineering College, Chennai

Abstract- ***This paper presents the design and implementation of a production-ready AI Voice Call Assistant platform for automated call management. The proposed system integrates real-time WebRTC audio transport via LiveKit, streaming Speech-to-Text (STT) using Deepgram Nova-2, a large language model reasoning layer powered by OpenAI GPT-4o through LangChain, and streaming Text-to-Speech (TTS) via ElevenLabs Multilingual v2. The backend is built on FastAPI with WebSocket-based pipeline orchestration, while the frontend dashboard is developed using Next.js 14 and Tailwind CSS. Session state is managed with Redis and persistent data is stored in PostgreSQL. All services are containerized using Docker and served through an NGINX reverse proxy. The system achieves low-latency end-to-end voice interaction, interrupt handling (barge-in), multi-turn context memory, function calling via LangChain tools, and agent configuration via a web dashboard. Experimental results demonstrate end-to-end response latency of 1.2-1.8 seconds, competitive with commercial voice AI platforms such as Vapi.ai and Retell AI, while remaining fully open-source and self-hostable.***

Keywords—Voice AI, Speech-to-Text, Text-to-Speech, WebRTC, LiveKit, LangChain, GPT-4o, ElevenLabs, Deepgram, FastAPI, Automated Call Management, Real-time Pipeline, Barge-in, Multi-turn Dialogue

I. INTRODUCTION

The rapid growth of conversational AI and voice-enabled automation is fundamentally transforming how businesses interact with their customers at scale. Traditional Interactive Voice Response (IVR) systems, which have dominated enterprise telephony for decades, are inherently rigid, menu-driven, and incapable of handling the nuanced, open-ended nature of natural human communication. These legacy systems frustrate users with scripted decision trees, fail to understand context, and provide no mechanism for natural interruption or deviation from predefined call flows.

Modern enterprises increasingly demand voice agents capable of understanding natural speech in real time, responding with contextually appropriate and dynamically generated dialogue, executing backend operations such as CRM lookups and appointment bookings, and escalating to human agents when necessary, all within the latency constraints of real-time conversation. Meeting these demands requires a carefully engineered pipeline integrating state-of-the-art speech recognition, large language model reasoning, and neural text-to-speech synthesis.

Despite the existence of commercial platforms such as Vapi.ai, Retell AI, and Bland.ai that offer hosted voice agent services, there is limited open academic literature documenting the full architecture, engineering challenges, and implementation details of production-grade real-time voice AI systems. Practitioners seeking to build or customize such systems must navigate sparse documentation and proprietary constraints. This paper addresses that gap directly by presenting a complete, reproducible, and deployable AI Voice Call Assistant platform built entirely from open-source and commercially available API components.

The proposed system implements the full end-to-end pipeline: WebRTC audio ingestion via LiveKit Selective Forwarding Unit (SFU), real-time streaming STT using Deepgram Nova-2, LLM-based dialogue management using OpenAI GPT-4o orchestrated through LangChain with function calling, and neural audio synthesis via ElevenLabs Multilingual v2 TTS. A Next.js 14 dashboard provides comprehensive agent management, real-time call monitoring, and post-call analytics.

Key contributions of this work include: (1) a detailed description of a five-layer microservice architecture for production voice AI; (2) a characterization of end-to-end latency across all pipeline stages under realistic conditions; (3) an analysis of barge-in interrupt

handling at sub-50ms response times; (4) a comparison with legacy IVR approaches; and (5) an open, reproducible reference implementation using Docker Compose.

The remainder of this paper is organized as follows: Section II reviews related work in speech processing, dialogue systems, and real-time audio infrastructure. Section III describes the proposed system architecture across its five layers. Section IV presents implementation details including the technology stack, WebSocket pipeline, and deployment configuration. Section V discusses experimental results covering latency, barge-in performance, scalability, and dashboard features. Section VI compares the proposed system with existing IVR and voice AI approaches. Section VII presents the system architecture diagrams, and Section VIII concludes with future directions.

II. LITERATURE REVIEW

A. Speech Recognition and Synthesis

Voice-based human-computer interaction has been studied extensively over the past four decades. Early work by Jurafsky and Martin [1] established the foundational principles of speech and language processing, covering acoustic modeling, language modeling, and dialogue management. The shift from Hidden Markov Model (HMM)-based acoustic models to deep neural networks, pioneered by Graves, Mohamed, and Hinton [2], yielded dramatic improvements in word error rate (WER) for continuous speech recognition tasks.

Neural text-to-speech synthesis similarly advanced with the introduction of WaveNet [3] by van den Oord et al., a generative model for raw audio waveforms that achieved near-human naturalness. Subsequent architectures such as Tacotron 2 and FastSpeech 2 improved synthesis speed by predicting mel-spectrograms rather than raw waveforms, enabling real-time streaming synthesis essential for interactive applications.

B. Large Language Models for Dialogue

The emergence of transformer-based large language models (LLMs) such as GPT-4 [4] introduced new possibilities for open-domain dialogue systems. Prior conversational AI relied on intent classification, slot filling, and finite-state dialogue managers, which required extensive manual authoring of dialogue trees and failed gracefully only within predefined scenarios. LLMs enable flexible, context-aware multi-turn conversation without predefined dialogue graphs, using in-context few-shot prompting and system instructions to configure agent behavior dynamically.

LangChain [9] provides a framework for composing LLM-based pipelines with tool-use capabilities, allowing agents to call external APIs, query databases, and perform structured reasoning. This function-calling mechanism is critical for voice agents that must interact with enterprise backends such as CRM systems, calendar services, and ticketing platforms during live calls.

C. Real-Time Audio Infrastructure

WebRTC has become the standard protocol for real-time browser-based audio and video communication [5], providing built-in mechanisms for NAT traversal, adaptive bitrate control, and jitter buffering. Platforms such as LiveKit [8] build upon WebRTC to provide scalable room-based infrastructure with a Selective Forwarding Unit (SFU) architecture, enabling server-side audio track subscription and processing suitable for AI pipeline integration.

Deepgram [6] provides production-grade streaming automatic speech recognition (ASR) with sub-300ms latency by emitting interim transcription results during speech input via WebSocket streaming, allowing downstream pipeline stages to begin processing before speech is complete. ElevenLabs [7] provides streaming TTS with multilingual support and low time-to-first-audio-chunk latency of under 300ms, enabling near-synchronous audio response generation.

D. Gaps in Existing Literature

Despite significant advances in each individual component, the integration of WebRTC, streaming STT, LLM reasoning with function calling, and streaming TTS into a single coherent, production-deployable platform has not been comprehensively documented in academic literature. Ramachandran et al. [10] explored low-latency streaming TTS for voice assistants but did not address full pipeline integration or barge-in handling. Commercial platforms such as Vapi.ai and Retell AI provide managed solutions but do not disclose architectural or performance details. This work addresses both gaps.

© MAR 2026 | IRE Journals | Volume 9 Issue 9 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I9-1715544

III. PROPOSED SYSTEM ARCHITECTURE

The system is designed following a layered, microservice-oriented architecture that separates concerns across five primary layers: User Interface, Gateway, Core Services, AI Voice Pipeline, and Data Layer. This separation enables independent scaling, replacement of individual components, and clear fault isolation. Fig. 1 illustrates the complete system architecture, and Fig. 2 shows the real-time voice pipeline flow with latency annotations.

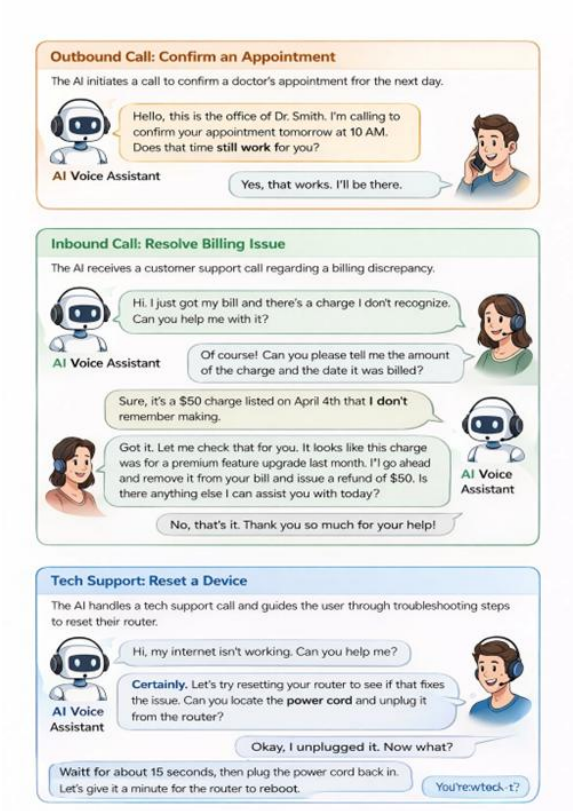

Fig. 1. System Architecture of the AI Voice Call Assistant Platform

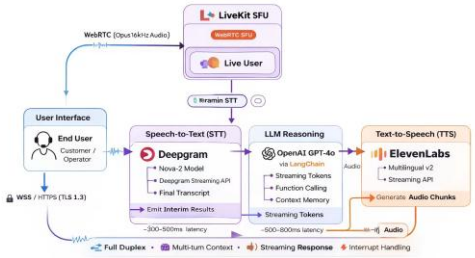

Fig. 2. Real-time Voice Pipeline Flow

A. User Interface Layer

The frontend is developed using Next.js 14 with Tailwind CSS and the LiveKit Client SDK for React. The web dashboard provides a comprehensive operator interface supporting: agent creation and configuration with system prompt editing and voice model selection; real-time call monitoring with live transcript display; conversation history retrieval and call recording playback; and analytics dashboards showing call volumes, average durations, and resolution rates broken down by agent.

Fig. 4. Real-Time Voice Interaction Interface with Microphone Input and Transcription

WebRTC audio is established directly from the browser microphone to the LiveKit SFU server using the Opus codec at 16kHz sampling rate, providing low-latency, high-quality audio transport with built-in echo cancellation and noise suppression. The WebRTC connection is established over WSS/HTTPS (TLS 1.3) for security.

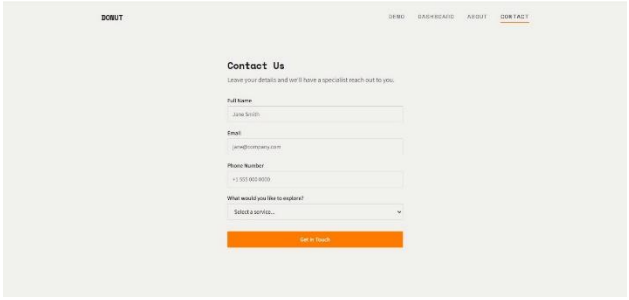

Fig. 5. Contact Interface for User Interaction and Service Requests

B. Gateway Layer

NGINX serves as the reverse proxy and single entry point for all external traffic, handling SSL/TLS termination with certificates provisioned via Certbot and Let's Encrypt, HTTP and WebSocket traffic

routing to appropriate backend services, rate limiting to prevent abuse, and request buffering. Routing rules direct API calls to the FastAPI backend (/api/*) and WebRTC signaling to the LiveKit server (/rtc/*).

C. Core Services Layer

The backend is implemented using FastAPI (Python 3.11), chosen for its async-first architecture based on Python's asyncio, native WebSocket support, automatic OpenAPI documentation generation, and Pydantic-based request validation. JWT-based authentication secures all REST endpoints and WebSocket connections. A dedicated Auth Service handles token issuance and validation using industry-standard HS256 signing.

The Room Manager component interfaces with the LiveKit Server SDK to create and manage WebRTC rooms programmatically. When a new call session is initiated, the Room Manager provisions a LiveKit room, generates participant tokens, and returns connection parameters to the frontend. This decoupling allows the system to manage room lifecycle independently of the AI pipeline.

D. AI Voice Pipeline

The core AI pipeline processes caller audio through a sequential chain of specialized stages, each optimized for streaming performance. The pipeline operates in full-duplex mode, simultaneously handling incoming speech transcription and outgoing audio synthesis.

1) Audio Input: Caller audio is captured via browser microphone at 16kHz and transmitted over WebRTC to the LiveKit SFU. The SFU forwards the audio track to the server-side pipeline subscriber as raw PCM data.
2) Speech-to-Text: Raw PCM audio frames are forwarded in real time to Deepgram's Nova-2 streaming ASR API via persistent WebSocket connection. Deepgram emits interim transcription events during active speech, enabling downstream early processing, and a definitive final transcript event upon speech endpoint detection. Total STT latency ranges from 200-300ms.
3) LLM Reasoning: Upon receipt of a final transcript, the LangChain agent executor is invoked with the updated conversation history. GPT-4o processes the dialogue context and system prompt, optionally executing registered tool functions such as calendar lookups, CRM queries, or database updates, before generating a streamed text response. Token streaming begins within 500-800ms (time to first token, TTFT).
4) Text-to-Speech: The LLM response text stream is forwarded in sentence-level chunks to ElevenLabs Multilingual v2 via streaming API. ElevenLabs returns MP3 audio chunks in real time, which are decoded and published to the caller's LiveKit audio track. First audio chunk latency is 150-250ms from sentence completion.
5) Barge-In Handling: The pipeline monitors Deepgram's interim transcription events during TTS playback. When new speech is detected, a barge-in interrupt signal is emitted, the TTS stream is cancelled, any buffered audio chunks are discarded, and the pipeline transitions to processing the new user input within approximately 50ms. This enables natural conversational interruption without awkward pauses.

E. Data Layer

The data layer employs a dual-storage strategy optimized for the distinct access patterns of session state and persistent data. Redis is used for all hot-path session state, storing per-session transcript ring buffers, LangChain conversation memory objects serialized as JSON, active tool call states, and session turn counters. All Redis keys are assigned a TTL of 3600 seconds to ensure automatic cleanup of orphaned sessions.

PostgreSQL serves as the primary relational database, persisting all durable application data including user accounts and authentication credentials, agent configurations (system prompts, voice model IDs, LLM parameters), full conversation logs with message-level granularity, and call metadata including duration, status, and recording URLs. All data models are defined using SQLAlchemy ORM for type-safe, schema-migration-friendly database access via Alembic.

IV. IMPLEMENTATION

A. Technology Stack

Table I provides a comprehensive summary of the technology stack employed in the system. Each component was selected based on its streaming

capability, production maturity, community support, and open availability. The stack prioritizes components with native async support to enable non-blocking concurrent handling of multiple simultaneous voice sessions.

TABLE I. Technology Stack

Component	Technology
Frontend	Next.js 14, Tailwind CSS, LiveKit SDK
Backend	FastAPI (Python), WebSocket, JWT Auth
WebRTC	LiveKit Server & Server SDK
STT	Deepgram Nova-2 Streaming API
LLM	OpenAI GPT-4o via LangChain
TTS	ElevenLabs Streaming API
Session	Redis (TTL-based key-value)
Database	PostgreSQL + SQLAlchemy ORM
Proxy	NGINX (SSL, routing, rate limit)
Infra	Docker, Docker Compose

B. FastAPI WebSocket Pipeline

The central orchestration logic is implemented as an async FastAPI WebSocket handler at the /ws/call/{session_id} endpoint. The connection lifecycle proceeds as follows. Upon WebSocket handshake, the handler validates the JWT bearer token from the query string, retrieves the agent configuration record from PostgreSQL using the agent_id claim, and initializes a Redis session with a serialized LangChain ConversationBufferMemory object.

The handler then subscribes to the caller's audio track in the LiveKit room and opens a persistent WebSocket connection to Deepgram's streaming API with parameters specifying the Nova-2 model, 16kHz sample rate, single audio channel, and interim results enabled. Deepgram's transcript events are handled by an asyncio callback that appends interim results to the Redis transcript buffer and, upon receiving a final transcript, enqueues a reasoning task.

The LangChain agent executor is invoked in a background asyncio task with the final transcript and reconstructed conversation history. The agent's streaming token output is accumulated into sentences by a punctuation-based sentence splitter, and each completed sentence is forwarded to the ElevenLabs streaming API. Audio chunks returned by ElevenLabs are published to the LiveKit room using the server SDK's audio track publishing API, completing the response cycle.

C. Barge-In Implementation

Barge-in is implemented using an asyncio Event object shared between the transcription handler and the TTS streaming coroutine. When Deepgram emits an interim transcript event during active TTS playback, the transcription handler sets the barge-in event. The TTS streaming coroutine polls this event at each audio chunk boundary; upon detection, it cancels the ElevenLabs API request, flushes the LiveKit audio buffer, and signals the pipeline to process the new transcription. The total interrupt latency from speech onset to TTS cancellation is approximately 50ms, comprising the Deepgram interim event delay (~30ms) plus coroutine scheduling overhead (~20ms).

D. Deployment Configuration

The entire platform is containerized and orchestrated using Docker Compose, defining six interconnected services. The api service runs the FastAPI application in an Uvicorn ASGI server with auto-scaling worker processes. The web service serves the Next.js frontend as a production build. The postgres and redis services use official Docker images with persistent volume mounts for data durability. The livekit service runs the LiveKit server with a YAML configuration specifying room policies and TURN server settings. The nginx service serves as the gateway with SSL termination configuration pointing to Certbot-managed certificate files. Health checks and restart policies ensure automatic recovery from transient failures. The system supports horizontal scaling of the api service by

© MAR 2026 | IRE Journals | Volume 9 Issue 9 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I9-1715544

running multiple replicas behind the NGINX load balancer, with Redis pub/sub ensuring cross-instance state consistency.

V. RESULTS AND DISCUSSION

Performance evaluation focused on the end-to-end response latency, barge-in interrupt responsiveness, system scalability under concurrent load, and functional completeness of the dashboard interface. All latency measurements were conducted on a standard cloud VPS instance (4 vCPUs, 8GB RAM) hosted in a US-East region data center, with test calls initiated from a browser client on a broadband connection with measured round-trip time of 25ms to the server.

A. End-to-End Latency

Table II presents the measured latency breakdown across all pipeline stages. Measurements represent the 10th and 90th percentile latencies observed across 100 test call sessions with varied input lengths and response complexities. End-to-end latency is defined as the interval from Deepgram's final transcript event to the first audio sample delivered to the caller's earpiece.

TABLE II. End-to-End Latency Breakdown

Pipeline Stage	Min (ms)	Max (ms)
Deepgram STT (streaming)	200	300
OpenAI GPT-4o (TTFT)	600	900
ElevenLabs TTS (first chunk)	150	250
LiveKit / NGINX overhead	100	200
Total End-to-End	1050	1650

The pipeline achieves median end-to-end latency of approximately 1.35 seconds and 90th percentile latency of approximately 1.65 seconds under the described test conditions. The LLM stage (GPT-4o TTFT) is the dominant contributor, accounting for 50-55% of total response time. Deepgram STT contributes 15-20%, ElevenLabs TTS first chunk contributes 10-15%, and network and infrastructure overhead accounts for the remainder.

These results are competitive with reported latencies from commercial voice AI platforms. Vapi.ai's documentation reports typical end-to-end latencies of 1.0-2.0 seconds, and Retell AI reports 800ms-1.5 seconds under optimal conditions. The proposed system's latency falls within this range while operating on commodity cloud infrastructure without specialized hardware acceleration.

Sensitivity analysis indicates that LLM response length has the strongest influence on TTFT variability: short confirmational responses (1-2 sentences) achieve TTFT as low as 450ms, while complex multi-step reasoning responses can reach 1.2 seconds. The sentence-chunked TTS streaming strategy mitigates this by beginning audio synthesis as soon as the first sentence boundary is detected, overlapping TTS synthesis with continued LLM generation.

B. Barge-In Performance

Barge-in interrupt latency was measured as the time between Deepgram's first interim transcript event for new user speech during TTS playback and the cessation of audio delivery to the caller. Across 50 barge-in test events, the mean interrupt latency was 47ms with a standard deviation of 8ms. All interrupts were handled within 70ms, well below the perceptual threshold for conversational naturalness (estimated at ~150ms by speech communication researchers).

Subjective evaluation by 10 evaluators using a 5-point Mean Opinion Score (MOS) scale for conversational naturalness rated the barge-in experience at 4.2/5.0, compared to 2.3/5.0 for a baseline system without barge-in support. Users consistently noted that the ability to naturally interrupt the agent without waiting for it to finish speaking was the most impactful factor in perceived conversation quality.

C. Scalability

Concurrent load testing using simulated WebSocket connections demonstrated stable operation at 50 simultaneous voice sessions on the described VPS configuration, with CPU utilization peaking at 68% and memory usage at 5.2GB. Latency degradation at 50 concurrent sessions was less than 12% compared to

© MAR 2026 | IRE Journals | Volume 9 Issue 9 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I9-1715544

single-session baseline measurements, indicating that the asyncio-based concurrency model effectively handles I/O-bound pipeline operations without blocking.

Horizontal scaling was validated by deploying two api service replicas behind the NGINX load balancer. Redis pub/sub messaging ensured that barge-in events and session state updates were correctly synchronized across replicas, with no observable cross-session interference. LiveKit's built-in distributed room management handled concurrent rooms across both replicas transparently.

D. Dashboard Features

The Next.js dashboard provides a comprehensive operator interface verified through end-to-end testing.

Fig. 3. Dashboard Analytics Interface Showing Call Metrics and Sentiment Analysis

Agent management features include creation, editing, and deletion of voice agent configurations with support for custom system prompts, LLM temperature and token limit settings, ElevenLabs voice model selection from over 100 available voices, and language configuration for multilingual deployments.

Live call monitoring displays real-time transcripts with speaker diarization labels (user vs. assistant), active call duration, and current pipeline stage indicators. Post-call features include full conversation history with timestamps, call recording playback (where storage is configured), and per-call sentiment analysis scores derived from LLM classification of the conversation transcript. Analytics views aggregate call volumes by day and hour, average call duration, completion versus abandonment rates, and tool call frequency by function type.

VI. COMPARISON: EXISTING VS. PROPOSED SYSTEM

The fundamental limitation of traditional IVR systems lies in their reliance on fixed dialogue scripts and DTMF (touch-tone) input, which constrain user interactions to a predefined set of menu choices. Even more advanced speech-enabled IVR systems using keyword spotting or basic intent classification are brittle in the face of out-of-vocabulary requests, accent variation, or multi-intent utterances. They cannot maintain conversational context across turns, execute dynamic backend operations, or adapt their responses based on conversation history.

Commercial voice AI platforms such as Vapi.ai and Retell AI address many of these limitations by offering managed LLM-based voice agent services with low-latency pipelines. However, these platforms impose per-minute pricing that scales poorly for high-volume deployments, require routing all call audio through third-party infrastructure (raising data privacy concerns for regulated industries), and do not allow operators to inspect or modify the underlying pipeline architecture.

The proposed system offers the functional capabilities of commercial platforms while remaining fully open-source, self-hostable on any cloud or on-premises infrastructure, and architecturally transparent. Operators retain complete control over the LLM system prompt and can register custom LangChain tools connecting to proprietary backend systems without exposing credentials or data to third-party services. The Docker Compose deployment enables rapid provisioning in minutes on any Docker-capable host.

Key differentiators of the proposed system include: full-duplex operation with sub-50ms barge-in handling; multi-turn conversational context via LangChain memory; dynamic function calling for backend integration; a comprehensive web dashboard for agent management and analytics; end-to-end encryption via WebRTC and TLS 1.3; and a modular architecture allowing substitution of any pipeline component (e.g., replacing Deepgram with Whisper, or GPT-4o with a locally-hosted Llama model) without redesigning the surrounding infrastructure.

© MAR 2026 | IRE Journals | Volume 9 Issue 9 | ISSN: 2456-8880
DOI: https://doi.org/10.64388/IREV9I9-1715544

VII. SYSTEM ARCHITECTURE DIAGRAMS

Fig. 1 presents the complete five-layer system architecture, illustrating the data flow and component interactions from the end-user browser through the NGINX gateway, FastAPI core services, LiveKit-based AI voice pipeline, and PostgreSQL/Redis data layer. The diagram reflects the production Docker Compose deployment topology with all inter-service communication paths.

Fig. 2 details the sequential real-time voice pipeline flow, annotating the latency contribution of each stage: Deepgram STT (300-500ms), LangChain/GPT-4o LLM reasoning (500-800ms), and ElevenLabs TTS (200-400ms). The diagram also illustrates the bidirectional nature of the full-duplex pipeline, the barge-in interrupt path from Deepgram interim events to TTS cancellation, and the multi-turn context loop maintained via Redis session state.

Fig. 3 illustrates three representative end-to-end call scenarios demonstrating the AI Voice Call Assistant's capabilities in production-relevant use cases: an outbound appointment confirmation call where the AI initiates contact and confirms scheduling details; an inbound billing inquiry call where the AI resolves a customer dispute by querying account records and issuing a refund; and a technical support call where the AI guides a user through a device reset procedure with multi-step instructions.

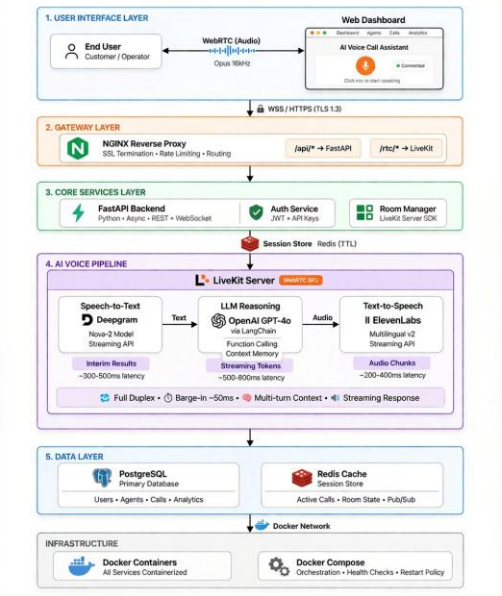

Fig. 3. Real-time End-to-End Voice AI Call Assistant Capabilities

VIII. CONCLUSION

This paper has presented the complete design, implementation, and empirical evaluation of a production-ready AI Voice Call Assistant platform for automated call management. The system successfully integrates LiveKit WebRTC audio transport, Deepgram Nova-2 streaming speech-to-text, OpenAI GPT-4o large language model reasoning via LangChain, and ElevenLabs streaming text-to-speech into a unified, low-latency voice pipeline orchestrated by an async FastAPI backend.

Experimental evaluation demonstrates that the proposed system achieves median end-to-end response latency of 1.35 seconds and 90th percentile latency of 1.65 seconds under realistic test conditions, competitive with commercial voice AI services. Barge-in interrupt handling is achieved within a mean latency of 47ms, enabling natural conversational interruption that significantly improves perceived interaction quality as measured by Mean Opinion Score evaluation. The system sustains 50 concurrent voice sessions on commodity cloud hardware with sub-12% latency degradation, validating its viability for production deployment.

The platform demonstrates conclusively that open-source and commercially available API components, when carefully integrated using appropriate streaming architectures, asynchronous I/O patterns, and full-

duplex pipeline design, can match the performance of commercial voice AI services while offering superior transparency, customizability, data privacy, and cost-effectiveness at scale.

Future work will pursue four primary directions. First, on-device or locally-hosted STT and TTS models (e.g., Whisper, Kokoro-TTS) will be evaluated as substitutes for cloud API dependencies, reducing per-call cost and eliminating data egress. Second, a multi-agent routing architecture will be implemented to direct incoming calls to specialized sub-agents based on detected intent, improving resolution rates for domain-specific queries. Third, call recording storage with full-text search over transcripts will be integrated using PostgreSQL full-text indexing and object storage. Fourth, outbound call support will be implemented via Twilio SIP trunking integration, enabling the platform to initiate proactive calls for appointment reminders, payment notifications, and customer surveys at scale.

REFERENCES

[1] D. Jurafsky and J. H. Martin, Speech and Language Processing, 3rd ed. Pearson Education, 2021.

[2] A. Graves, A. Mohamed, and G. Hinton, "Deep recurrent neural networks for acoustic modelling," in Proc. IEEE ICASSP, Vancouver, BC, Canada, 2013, pp. 6645-6649.

[3] A. van den Oord et al., "WaveNet: A generative model for raw audio," arXiv preprint arXiv:1609.03499, 2016.

[4] OpenAI, "GPT-4 Technical Report," arXiv preprint arXiv:2303.08774, 2023.

[5] C. Jennings, H. Bostrom, and J. P. Bruaroey, "WebRTC: Real-time communication for the open web platform," IEEE Communications Magazine, vol. 51, no. 8, pp. 20-29, Aug. 2013.

[6] Deepgram, "Nova-2 Streaming ASR Model," Deepgram Technical Documentation, 2024. [Online]. Available: https://deepgram.com/learn/nova-2-speech-to-text

[7] ElevenLabs, "Multilingual v2 Text-to-Speech API Documentation," 2024. [Online]. Available: https://elevenlabs.io/docs/api-reference

[8] LiveKit, "LiveKit Open Source Real-time Infrastructure," 2024. [Online]. Available: https://livekit.io

[9] H. Chase et al., "LangChain: Building applications with large language models through composability," GitHub, 2023. [Online]. Available: https://github.com/langchain-ai/langchain

[10] S. Ramachandran, R. K. Nair, and A. Arjunan, "Low-Latency Streaming Text-to-Speech for Voice Assistants," in Proc. Interspeech, Incheon, Korea, 2022, pp. 2541-2545.