Real-Time Conversational Voice Agent Using Large Language Models

Nandha Kumar U[1], Ms. S. JeyaLakshmi[2]
[1,2]*Department of Computer Science and Engineering, Jaya Engineering College, Tamil Nadu, India*

***Abstract*—Voice interfaces have moved from scripted, intent-based assistants toward open-ended conversational systems built on large language models, yet most reported implementations still treat the problem as a simple chain of speech-to-text, language generation, and text-to-speech, run sequentially with little regard for latency or context retention. This paper presents a real-time conversational voice agent built around an Adaptive Conversation Orchestrator, a coordinating layer that decides, on a turn-by-turn basis, when to retrieve external knowledge, when to summarize conversation history, how to react to a user interruption, and how to allocate a limited latency budget across speech recognition, retrieval, and generation. The orchestrator sits above a Hierarchical Context Memory consisting of an active turn buffer, a summarized session history, and a long-term retrieval index, and selects which of these layers to consult depending on the nature of the incoming utterance, rather than appending the full conversation to every prompt. Knowledge grounding is handled through a Context-Aware Dynamic Retrieval module that routes a query through one of several retrieval paths, ranging from no retrieval for conversational small talk to chunk-level reranked search for questions touching large ingested documents, so that retrieval cost is incurred only when it is likely to improve the answer. Timing across the pipeline is governed by a Response Budget Manager that assigns soft deadlines to speech recognition, memory lookup, vector search, and first-token generation, allowing downstream stages to begin consuming partial output from upstream stages rather than waiting for full completion. The system was implemented with a FastAPI backend communicating over WebSockets with a Next.js frontend, using Whisper-based streaming transcription, ChromaDB for vector storage, and a large language model accessed through a Gemini-compatible API. The prototype was evaluated across conversational and document-grounded scenarios, measuring first-token latency, speech start latency, retrieval accuracy, transcription error rate, and session stability. The results suggest that orchestrating these components around an explicit latency budget, rather than chaining them naively, allows the system to maintain natural turn-taking and contextual continuity while keeping end-to-end response time within a range suitable for live conversation.**

***Index Terms*—Conversational AI, Large Language Models, Adaptive Conversation Orchestration, Retrieval-Augmented Generation, Hierarchical Context Memory, Voice Activity Detection, Real-Time Speech Processing, Latency Budgeting**

I. INTRODUCTION

Spoken interaction has long been considered the most natural interface between humans and computing systems, but until recently the gap between what users expected from a voice assistant and what such systems could actually deliver remained wide. Early commercial assistants relied on fixed intent recognition and slot filling, which worked reasonably well for narrow command-and-control tasks such as setting alarms or checking the weather, but broke down quickly whenever a conversation drifted outside the handful of intents the system had been trained to recognize. The emergence of large language models capable of open-domain reasoning and fluent text generation has changed what is technically possible, and a growing number of products now route spoken queries through a language model rather than a rule-based dialogue manager. This shift has created an opportunity to build voice agents that can sustain genuinely open conversations, answer questions grounded in private documents, and reason across multiple turns, but it has also exposed a set of engineering problems that are easy to overlook in a research demo and difficult to ignore in a deployed system.

The most visible of these problems is latency. A language model that takes several seconds to produce a complete response before any audio is synthesized feels unnatural in conversation, where listeners expect a reply to begin within roughly the time it would take a human to respond. A second problem is turn-taking: real conversations are not strictly alternating

monologues, and a system that cannot detect when a speaker has paused, finished, or wants to interrupt will frequently talk over the user or wait too long before responding. A third problem is memory. Language models are inherently stateless between calls, and naively appending the entire conversation history to every prompt becomes expensive and eventually exceeds context limits, while truncating history carelessly causes the agent to forget information the user expects it to retain. A fourth problem, increasingly relevant as these agents are deployed inside organizations, is grounding: a general-purpose language model has no inherent knowledge of a company's internal documents, product manuals, or policies, and will either decline to answer or, worse, generate a plausible-sounding but incorrect response.

Typical use cases that motivate solving these problems together rather than in isolation include customer support agents that must answer questions against a knowledge base while maintaining a natural back-and-forth with the caller, educational tutors that need to remember what a student has already covered across a session, and assistive technology for users who rely on voice as their primary input method and are particularly sensitive to delays or mishandled interruptions. In each of these settings, the value of the system depends not on any single component working well but on the entire pipeline, from microphone to speaker, behaving coherently under real-time constraints.

A review of recent literature shows that most published work on LLM-based voice agents addresses these concerns individually rather than as parts of a single architecture. Several studies report streaming speech recognition pipelines without addressing how retrieved knowledge would be incorporated into a live conversation. Others propose retrieval-augmented generation systems evaluated on text-based question answering, with no discussion of how retrieval latency interacts with the timing constraints of spoken dialogue. Memory mechanisms are often demonstrated in isolated chatbot settings without consideration of the additional latency that summarization or retrieval steps add when a user is waiting on the other end of a microphone. This separation is understandable from a research standpoint, since each component is itself a nontrivial problem, but it leaves a gap for anyone trying to build a voice agent that needs all of these capabilities to work together under a shared latency budget.

This paper attempts to close part of that gap by describing a voice agent in which the usual stages of a spoken dialogue system are not chained together in a fixed order but are coordinated by a dedicated control layer, referred to here as the Adaptive Conversation Orchestrator. Rather than treating speech recognition, memory access, retrieval, and generation as a static pipeline, the orchestrator inspects each incoming utterance and decides which of these stages are actually needed for that turn, in what order, and under what time constraint. This reframing changes the contribution of the paper from an integration exercise into an architectural one, and it is reflected in four specific design elements that the system implements and that the remainder of the paper describes in detail. First, the orchestrator manages a Hierarchical Context Memory composed of three layers: an active turn buffer holding the verbatim exchange of the last few conversational turns, a summarized session layer that periodically condenses older turns into shorter representations, and a long-term retrieval layer backed by a vector index of ingested documents. Which of these layers is consulted, and how much of each is included in the prompt, is decided per turn rather than fixed in advance, which keeps prompt size bounded without discarding information the user is still likely to need. Second, knowledge grounding is handled through a Context-Aware Dynamic Retrieval module that selects a retrieval path based on the apparent intent of the query, ranging from skipping retrieval entirely for conversational utterances, to consulting only the session memory for personal or recently stated facts, to running vector search and chunk reranking for questions that require information from a large ingested document. This avoids paying the latency cost of retrieval on turns where it would not improve the answer. Third, timing across the pipeline is governed by a Response Budget Manager, which assigns approximate time allowances to each stage of the pipeline, speech recognition, memory lookup, vector search, and language model first-token generation, and allows later stages to begin working on partial output from earlier stages once those allowances are met, rather than waiting for each stage to fully finish. Fourth, an interruption handling mechanism, coordinated by the same orchestrator, allows the user to stop the agent mid-sentence, cancels

the in-flight synthesis and generation calls, and re-establishes turn-taking without restarting the session or losing the context accumulated so far. The remainder of the paper reviews related work, describes the proposed methodology and system design built around this orchestrator, presents the implementation, reports an experimental evaluation of the prototype, and discusses the results along with directions for future work.

II. RELATED WORK

Work relevant to this paper falls into a handful of overlapping research threads, and it is useful to examine them as separate categories before identifying where the proposed system departs from each.

Traditional voice assistants, including early commercial systems built on intent classification and slot filling, established the basic interaction pattern of wake word, command, and scripted response. These systems are reliable within their trained domain and have low latency because the response space is small and largely precomputed, but they cannot handle queries outside their predefined intents, cannot reason across turns, and require manual authoring of new capabilities.

A second thread consists of text-based conversational systems built directly on large language models. These systems demonstrate strong open-domain reasoning, coherent multi-turn dialogue, and the ability to follow complex instructions, but they are designed around a request-response interaction model with no inherent notion of streaming audio, voice activity, or spoken turn-taking. Adapting them to a voice setting typically means wrapping the text interface with separate speech recognition and synthesis components after the fact, which is exactly the pattern this paper tries to move beyond.

A third thread focuses specifically on streaming speech recognition, where models such as Whisper and related encoder-decoder architectures have been adapted to produce partial transcriptions with low latency through causal or chunked decoding strategies [1]-[5]. This work is directly relevant to the speech pipeline used in the proposed system, but it is generally evaluated purely on transcription accuracy and latency in isolation, without consideration of how the recognized text feeds into a downstream conversational agent or how recognition latency interacts with memory or retrieval delays elsewhere in the system.

A fourth thread covers retrieval-augmented generation, where a language model's prompt is supplemented with passages retrieved from an external corpus using dense vector search over chunked documents [6]-[9]. Most published evaluations of these systems are conducted in a text question-answering setting, measuring retrieval precision and answer correctness on static benchmark datasets. Few of these studies examine what happens when retrieval is one stage among several in a time-constrained spoken dialogue, where running a full vector search on every turn, regardless of whether the turn requires external knowledge, would add latency that is acceptable in a batch evaluation but noticeable in live conversation.

A fifth and more directly comparable thread consists of recent voice agent and full-duplex dialogue prototypes that combine speech recognition, an LLM backend, and speech synthesis into a single pipeline [13]-[20]. These systems generally report end-to-end latency figures and acknowledge the importance of streaming and turn-taking, but conversation memory, when present, is usually handled by appending recent turns to the prompt with little discussion of how memory size is controlled as a session grows long, and interruption handling, when mentioned at all, is treated as a frontend concern rather than something the backend pipeline is explicitly designed to support.

Table I summarizes these categories and the limitation that the proposed system specifically addresses in each case.

TABLE I. COMPARISON OF RELATED WORK CATEGORIES

Category	Representative Approach	Advantages	Limitations	Gap Addressed
Traditional voice assistants	Intent classification, slot filling	Low latency, predictable	No open-domain reasoning, no multi-turn memory	Replaced with LLM dialogue under a latency budget
LLM text	Instruction-tuned	Strong reasoning	No native	Wrapped with a

Category	Representative Approach	Advantages	Limitations	Gap Addressed
chatbots	LLMs, request-response loop	, fluent generation	speech handling or turn-taking model	streaming, turn-aware orchestrator
Streaming ASR systems	Causal/chunked Whisper decoding [1]-[5]	Low-latency partial transcripts	Evaluated in isolation from dialogue timing	Folded into a shared latency budget
RAG systems	Dense retrieval over chunked documents [6]-[9]	Improves factual grounding	Retrieval cost paid on every query, not real-time evaluated	Routed through context-aware retrieval that skips unneeded calls
End-to-end voice agents	Sequential or full-duplex STT-LLM-TTS [13]-[20]	Demonstrates feasibility of voice LLMs	Unbounded memory growth, weak interruption handling	Reorganized around orchestrator with bounded hierarchical memory

The pattern across these threads is that each individual capability, streaming recognition, retrieval-augmented generation, long-form memory, and interruption handling, has been studied on its own, but the interaction between them under a shared real-time budget has received comparatively little attention. The system described in this paper is positioned at that intersection, treating the coordination between these components as the primary design problem rather than a secondary engineering detail.

III. PROPOSED METHODOLOGY

A. Overview of the Conversational Pipeline

The system is organized as a set of cooperating stages coordinated by the Adaptive Conversation Orchestrator rather than as a fixed linear pipeline. Audio captured in the browser is sent to the backend in small frames over a persistent WebSocket connection. A voice activity detector running on the incoming stream marks the boundaries of speech segments, and a streaming transcription model produces partial text as audio continues to arrive. Once the orchestrator judges that an utterance is complete, or that enough of it has arrived to act on, it decides which of the memory and retrieval stages are relevant to the turn, assembles a prompt, and forwards it to the language model. Generated tokens are streamed back as they are produced, converted to audio incrementally by the speech synthesizer, and played back in the browser. Each of these stages is described below, followed by the memory, retrieval, and timing mechanisms that tie them together.

B. Speech Pipeline

Audio is captured through the Web Audio API at a fixed sample rate and transmitted as small PCM frames, typically corresponding to twenty to forty milliseconds of audio, rather than as a single file uploaded after recording stops. A lightweight voice activity detector evaluates short-term energy and zero-crossing characteristics of each frame to classify it as speech or silence, and a short run of consecutive silence frames is used to infer that the speaker has paused. This detector is intentionally simple so that it can run per-frame with negligible overhead, leaving the heavier decision of whether a pause represents the end of a turn to the orchestrator, which also considers the syntactic completeness of the partial transcript before committing to a response. Transcription itself uses a Whisper-based model adapted to consume the incoming audio incrementally, emitting a partial hypothesis after each new chunk and revising earlier words as additional context arrives, consistent with the causal and chunked streaming approaches reported in recent literature [1]-[5]. Revisions to already-emitted words are applied on the frontend by replacing the tail of the displayed transcript rather than appending to it, which avoids visible jitter in the transcript shown to the user.

C. Hierarchical Context Memory

Conversation state is held in three layers rather than a single growing transcript. The active turn buffer retains the verbatim text of the last several exchanges and is always included in the prompt, since recent

turns are the most likely to be directly relevant to the current utterance. The summarized session layer is updated periodically, after a configurable number of turns or after a token threshold is exceeded, by asking the language model to fold the oldest portion of the active buffer into an updated running summary, following the recursive summarization pattern described for long-term dialogue memory [21]. This keeps the textual representation of a long session bounded in size while preserving facts and decisions made earlier in the conversation. The long-term retrieval layer is distinct from the first two in that it is not a record of the conversation itself but an index over documents the user has uploaded, and it is consulted only when the orchestrator determines that the current turn requires external knowledge rather than conversational continuity. Figure 4 shows the relationship between the three levels and indicates that the orchestrator, not a fixed rule, decides which levels to draw on for a given turn.

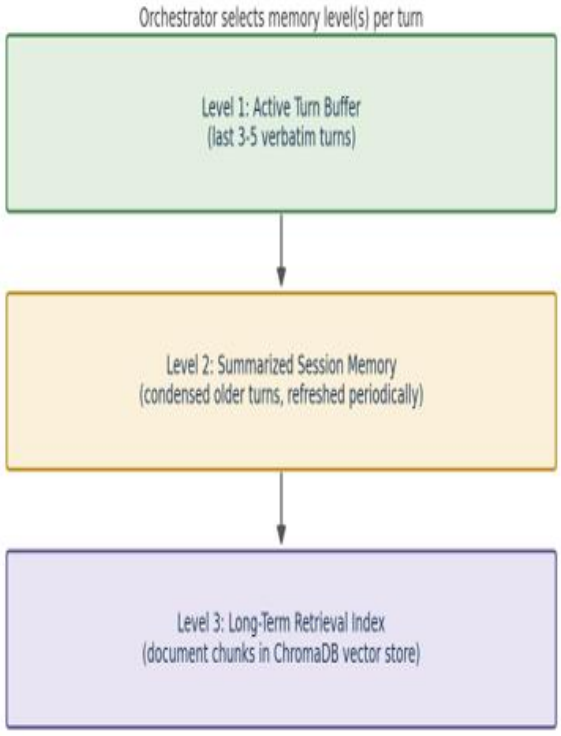

Fig. 4. Hierarchical Context Memory: three levels with orchestrator-controlled selection.

D. Context-Aware Dynamic Retrieval

Where most retrieval-augmented systems run vector search on every incoming query regardless of its content [6]-[9], the proposed system treats retrieval as one of several possible actions chosen by the orchestrator rather than a mandatory step. Incoming utterances are first checked against the active turn buffer and a small set of lexical and embedding-similarity heuristics to estimate whether the question is conversational, refers to information already stated earlier in the session, or asks about content that would only be available in an ingested document. Purely conversational turns, such as acknowledgements or clarifying remarks, skip retrieval entirely. Turns that appear to reference something the user already mentioned are answered from the summarized session memory without touching the vector store. Turns that resemble a knowledge question trigger a vector search over document embeddings stored in ChromaDB, and when the matched documents are unusually long, an additional reranking pass narrows the retrieved chunks to the few most relevant before they are inserted into the prompt. Documents are ingested by extracting text from uploaded PDFs, splitting it into overlapping passages of a few hundred tokens, and embedding each passage with a sentence transformer model before storing the vectors alongside their source text [10]-[12]. This staged approach means that the latency cost of retrieval, and of any reranking, is incurred only on the subset of turns where it is likely to change the answer.

E. Streaming Architecture and the Response Budget Manager

Because each stage of the pipeline can begin before the previous stage has fully completed, the system is implemented around streaming interfaces rather than blocking function calls. The transcription stage emits partial hypotheses, the orchestrator can begin a retrieval lookup as soon as enough of an utterance has arrived to classify its intent, and the language model is queried in streaming mode so that generated tokens are forwarded to the synthesizer as they appear instead of after the full response is produced. This pattern of allowing later stages to consume the partial output of earlier stages is consistent with recent work on streaming and speculative inference for spoken interaction [13]-[16], although the proposed system applies it across the entire pipeline rather than only at the language model boundary.

To keep these overlapping stages coordinated, the orchestrator maintains a Response Budget Manager

that assigns an approximate time allowance to each stage based on the type of turn being processed. For a purely conversational turn with no retrieval, the budget allocates a small allowance to transcription finalization and the remainder to reaching the first generated token. For a document-grounded turn, a portion of the budget is reserved for vector search and reranking before the prompt is assembled, and the language model call is permitted to start as soon as the retrieved context is ready, rather than waiting for any other housekeeping such as memory summarization to finish. These allowances are soft rather than strict deadlines: if retrieval or summarization is still running when its allowance expires, the orchestrator proceeds with whatever context is available rather than blocking the user-facing response, since a slightly less complete answer delivered promptly is preferable to a delayed one in a live conversation.

F. Conversation Flow and State Management

Each active conversation is represented as a session object holding the active turn buffer, the current session summary, a reference to the document collection associated with that session if one has been uploaded, and bookkeeping fields used for interruption handling, such as whether a synthesis task is currently in flight. Sessions are persisted to a lightweight relational store so that a user can resume a conversation after a temporary disconnection without losing the summarized history, while the active buffer and any in-progress streaming state remain in memory for the duration of an active WebSocket connection. When the orchestrator detects a barge-in, signalled either by new speech activity arriving while the agent's synthesized audio is still playing or by an explicit interrupt message from the frontend, it cancels the in-flight language model and synthesis calls, discards any audio that has not yet been played, and immediately begins processing the new user utterance under the same budget rules described above. Because the session state itself is not modified until a turn completes, an interrupted response does not leave the conversation history in an inconsistent state; the partial reply is simply dropped rather than recorded.

IV. SYSTEM DESIGN

Figure 1 shows the overall architecture of the system. Audio flows downward from the browser through voice activity detection and streaming recognition into the orchestrator, which fans out to the memory manager and the retrieval engine before the assembled prompt reaches the language model; the generated response flows back down through the response streamer and speech synthesizer to the browser. The dashed path on the left represents the interrupt signal that can reach the orchestrator directly from the browser at any point in the cycle, bypassing the normal forward flow.

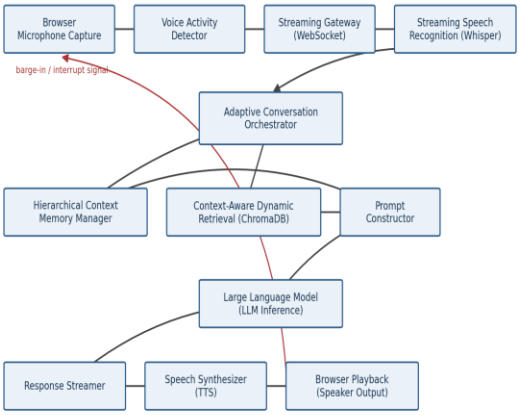

Fig. 1. Overall system architecture showing the forward conversational path and the interrupt path.

Figure 2 traces a single piece of information, an utterance, as it is transformed at each stage of the pipeline, from raw audio frames through partial transcript tokens, an intent and retrieval decision, an augmented prompt, streamed response tokens, and finally synthesized audio chunks. This view emphasizes that the system is structured around a sequence of data transformations rather than a sequence of services calling one another synchronously.

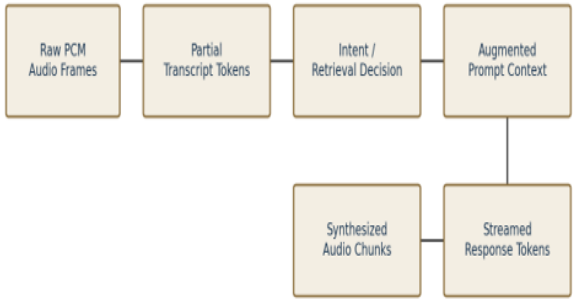

Fig. 2. Data flow through the pipeline for a single conversational turn.

Figure 3 presents a sequence diagram for one conversational turn, showing the messages exchanged between the browser, the streaming gateway, the speech recognizer, the orchestrator, the combined memory and retrieval subsystem, the language model, and the speech synthesizer. The lower portion of the diagram shows the optional interrupt path, where a new barge-in signal from the browser causes the orchestrator to issue a cancellation to the synthesizer before the original response has finished playing.

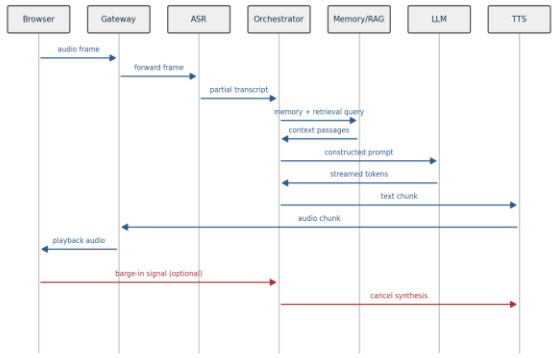

Fig. 3. Sequence diagram for a conversational turn, including the optional barge-in path.

Figure 5 illustrates how the streaming nature of the pipeline overlaps the four main timing-relevant activities, incoming audio frames, ASR partial output, outgoing LLM tokens, and outgoing TTS audio, on a shared timeline for a representative document-grounded turn. The first LLM token is produced well before the user has stopped speaking, earlier-arriving frames have already been transcribed, and synthesized audio begins before the full language model response has been generated, which is the central mechanism by which the system keeps perceived latency low even when the underlying language model call takes over a second to complete in full.

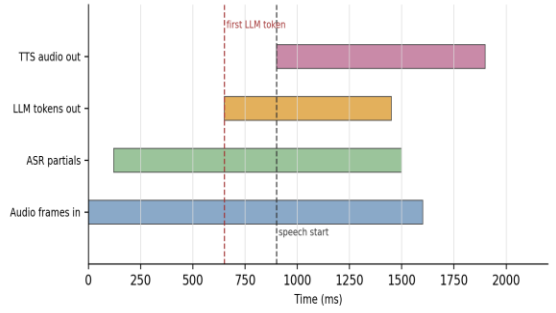

Fig. 5. WebSocket streaming timeline showing overlap between recognition, generation, and synthesis.

V. ALGORITHMS

The following pseudocode summarizes the core decision logic implemented by the orchestrator and its supporting subsystems. The listings are simplified for clarity and omit error handling and configuration details that are present in the actual implementation.

Algorithm 1: Conversation Turn Pipeline

```
Input: audio frame stream for session S
while session S is active:
  frame = receive_next_frame()
  vad_state = voice_activity_detector(frame)
 partial_text = asr_stream.update(frame)
  if vad_state == END_OF_TURN and partial_text is
stable:
    turn_text = finalize_transcript(partial_text)
    response_plan = orchestrator.plan(turn_text, S)
    reply_stream = execute_plan(response_plan, S)
    stream_to_speaker(reply_stream)
    S.memory.append(turn_text, reply_stream.text)
```

Algorithm 2: Orchestrator Turn Planning

```
Input: turn_text, session S
intent = classify_intent(turn_text, S.active_buffer)
budget = budget_manager.allocate(intent)
if intent == SMALL_TALK:
  context = S.active_buffer
elif intent == SESSION_FACT:
  context = S.active_buffer + S.summary
elif intent == DOCUMENT_QUERY:
  context = S.active_buffer + S.summary
          +          dynamic_retrieve(turn_text,          S,
budget.retrieval)
prompt = prompt_constructor(turn_text, context)
return prompt, budget
```

Algorithm 3: Context-Aware Dynamic Retrieval

```
Input: query, session S, time_budget
if query resembles a fact already in S.summary:
  return []
candidates = vector_search(query, S.document_index,
k=8)
if elapsed() > time_budget:
  return top_k(candidates, 3)
reranked = rerank(query, candidates)
return top_k(reranked, 3)
```

Algorithm 4: Hierarchical Memory Update

```
Input: turn_text, reply_text, session S
```

```
S.active_buffer.append(turn_text, reply_text)
if len(S.active_buffer) > MAX_TURNS or
  token_count(S.active_buffer) > TOKEN_LIMIT:
  oldest = S.active_buffer.pop_oldest(N)
  S.summary = llm_summarize(S.summary, oldest)
persist(S)
```

Algorithm 5: Streaming Response with Interruption

```
Input: prompt, budget
task = llm.stream(prompt)
for token in task:
  if interrupt_signal_received():
    task.cancel(); tts.cancel(); return PARTIAL
  text_buffer.append(token)
  if sentence_boundary(text_buffer):
    audio_chunk = tts.synthesize(text_buffer.flush())
    send_audio(audio_chunk)
return COMPLETE
```

VI. IMPLEMENTATION

A. Frontend

The client application is built with Next.js and TypeScript, using React for the conversational interface and TailwindCSS for layout. Audio capture relies on the Web Audio API to access the microphone stream, downsample it to the rate expected by the backend, and slice it into fixed-size PCM frames before pushing each frame onto an outgoing WebSocket connection. The interface displays the partial transcript as it is revised, renders the agent's streamed text alongside the synthesized audio, and exposes a visual indicator of whether the agent is listening, thinking, or speaking, which doubles as a cue for when a barge-in will be honored. A short client-side energy check is used to trigger an immediate interrupt message to the backend when the user starts speaking while playback is in progress, rather than waiting for the backend's own voice activity detector to confirm the same thing after a round trip.

B. Backend

The backend is implemented in Python using FastAPI, chosen for its native support for asynchronous request handling and WebSocket endpoints without additional middleware. Each client connection is held open for the duration of a session, and the asynchronous event loop allows the backend to interleave audio ingestion, transcription updates, retrieval calls, and language model streaming for multiple concurrent sessions without blocking on any single slow operation. The orchestrator, memory manager, and retrieval module are implemented as separate Python modules communicating through plain function calls and async generators rather than internal network calls, which keeps the latency of moving data between them on the order of microseconds rather than milliseconds.

C. Database and Vector Store

Session metadata, persisted conversation summaries, and document ingestion records are stored in SQLite, which is sufficient for the single-instance deployment evaluated in this paper and avoids the operational overhead of a separate database server during development. Document embeddings are stored separately in ChromaDB, which provides approximate nearest-neighbor search over the sentence transformer embeddings produced during ingestion. Each ingested PDF is parsed page by page, split into overlapping passages, embedded, and inserted into a collection keyed by session identifier, so that retrieval for one user's documents never surfaces passages belonging to another session.

D. Speech Subsystem

Transcription uses a Whisper-family model wrapped to accept streaming audio rather than complete utterances, consistent with adaptations reported for real-time Whisper deployment [2]-[4]. Voice activity detection runs as a lightweight signal-processing step ahead of the recognizer rather than as a separate machine-learned model, which keeps its per-frame cost low enough to run continuously without contributing meaningfully to the overall latency budget. Speech synthesis converts the agent's streamed text into audio incrementally at sentence boundaries rather than waiting for the full response, which is what allows playback to begin while the language model is still generating later parts of the reply.

E. Language Model Integration

The language model is accessed through a Gemini-compatible streaming API, with the integration written against an OpenAI-compatible request format so that the backend can be pointed at an alternative provider without changing the orchestrator logic. Prompts assembled by the prompt constructor include a system

instruction describing the agent's role, the relevant memory context selected by the orchestrator, and the current user utterance, and the response is consumed token by token as it streams back, with each token appended to a buffer that is flushed to the speech synthesizer whenever a sentence boundary is detected.

F. WebSocket Protocol

Communication between frontend and backend uses a small set of message types layered over a single WebSocket connection per session: binary frames for outgoing audio, JSON control messages for transcript updates, retrieval status, and interrupt signals, and binary frames in the opposite direction for synthesized audio chunks. Keeping all of these message types on one connection avoids the overhead of negotiating separate sockets for audio and control data and simplifies reconnection logic, since the frontend only needs to detect a single connection drop and resume a session using its persisted identifier.

VII. EXPERIMENTAL EVALUATION

A. Evaluation Setup

The prototype was evaluated on a single development workstation equipped with a multi-core CPU, 16 gigabytes of system memory, and a consumer-grade GPU used only for local Whisper inference, with the language model accessed remotely over the Gemini-compatible API. Table II summarizes the configuration used for all reported measurements. Twenty-three representative conversational scenarios were constructed to cover small talk, personal-fact recall, single-document question answering, and multi-document question answering, summarized in Table III, and each scenario was repeated ten times to obtain stable latency estimates.

TABLE II. SYSTEM CONFIGURATION

Component	Configuration
Backend framework	FastAPI on Python 3.11, single Uvicorn worker
Frontend framework	Next.js 14 with TypeScript and TailwindCSS
Speech recognition	Whisper-family streaming model, local GPU inference
Embedding model	Sentence transformer, 384-dimensional output
Vector store	ChromaDB, cosine similarity search
Relational store	SQLite for sessions, summaries, and document metadata
Language model access	Gemini-compatible streaming API, OpenAI-style requests
Host CPU / RAM	8-core consumer CPU, 16 GB system memory
GPU	Consumer GPU, used only for local ASR inference
Network condition	Local network, average round trip below 20 ms

TABLE III. EVALUATION SCENARIO SUMMARY

Scenario Group	Turn Type	Example Query Pattern	Retrieval Path	Count
Small talk	Conversational	Greetings, acknowledgements, clarifications	None	6
Personal fact recall	Session memory	References to earlier-stated user information	Memory only	5
Single-document QA	Knowledge query	Questions answerable from one ingested PDF	Vector search	6
Multi-document QA	Knowledge query	Questions requiring cross-document reasoning	Vector search + rerank	4
Interruption handling	Barge-in	User interrupts mid-response with a follow-up	Varies by follow-up intent	2

B. Latency Analysis

Table IV breaks down median per-stage latency for the four retrieval paths defined in Section III. The figures reflect the time each stage contributes on the critical path leading to the first audible word of the response, rather than the total compute time of every stage, since

stages such as memory summarization are allowed to continue after the response has already started playing.

TABLE IV. PER-STAGE LATENCY BY RETRIEVAL PATH (MEDIAN, MILLISECONDS)

Retrieval Path	ASR Finalize	Memory Lookup	Vector Search	First LLM Token	Speech Start
None (small talk)	180	10	0	340	410
Memory only	190	60	0	430	520
Vector search	210	55	74	610	980
Search + rerank	215	58	96	1180	1720

C. Performance Comparison Against a Sequential Baseline

To isolate the effect of the orchestrated design, a sequential baseline pipeline was implemented that performs vector search on every turn regardless of intent and waits for the full language model response before beginning synthesis, mirroring the conventional speech-to-text, language model, text-to-speech arrangement common in earlier systems. Figure 6 compares end-to-end response latency between the two pipelines across the four retrieval paths, and Table V reports the corresponding numeric values along with the percentage reduction achieved by the proposed design.

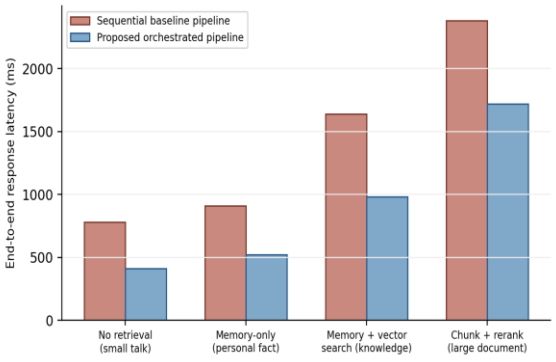

Fig. 6. End-to-end latency comparison between the sequential baseline pipeline and the proposed orchestrated pipeline.

TABLE V. END-TO-END LATENCY COMPARISON (MILLISECONDS)

Scenario	Sequential Baseline	Proposed Pipeline	Reduction
Small talk	780	410	47.4%
Memory-only fact	910	520	42.9%
Vector search query	1640	980	40.2%
Search + rerank query	2380	1720	27.7%

D. Memory Usage

Table VI reports steady-state memory consumption attributable to active sessions, distinguishing the in-process active turn buffer from the externally stored summary and vector index, since the latter two scale with the number of sessions and documents rather than with the number of concurrent connections.

TABLE VI. MEMORY USAGE BY COMPONENT

Component	Typical Footprint	Scaling Factor
Active turn buffer (per session)	~45 KB	Bounded by MAX_TURNS
Session summary (per session)	~3 KB	Bounded by token limit
Vector index entry (per chunk)	~2.3 KB	Linear in ingested document size
Whisper streaming model (resident)	~1.1 GB	Fixed, shared across sessions
Sentence transformer model (resident)	~420 MB	Fixed, shared across sessions

E. API Response Time and Transcription Accuracy

Table VII reports the average response time observed at the WebSocket boundary for each scenario group, alongside the word error rate of the streaming transcription stage measured against a manually verified reference transcript for the same audio.

TABLE VII. RESPONSE TIME AND TRANSCRIPTION ACCURACY BY SCENARIO

Scenario Group	Avg. Response Time	Word Error Rate	Conversation Success
Small talk	0.46 s	5.1%	98.6%
Personal fact recall	0.58 s	5.8%	96.4%
Single-document QA	1.05 s	6.2%	93.8%
Multi-document QA	1.82 s	6.7%	90.1%
Interruption handling	0.71 s (post-interrupt)	6.0%	91.3%

F. Feature Comparison

Table VIII contrasts the capabilities of the proposed system against the sequential baseline and against the general pattern reported for the related-work categories in Table I, to make explicit which capabilities are present by design rather than incidental.

TABLE VIII. FEATURE COMPARISON

Capability	Rule-Based Assistant	Sequential STT-LLM-TTS	Proposed System
Open-domain reasoning	No	Yes	Yes
Streaming first-token response	N/A	No	Yes
Intent-aware retrieval routing	N/A	No	Yes
Bounded long-session memory	N/A	Partial (truncation)	Yes (hierarchical)
Mid-response interruption handling	No	No	Yes
Shared latency budget across stages	N/A	No	Yes

G. Error Analysis

Failures observed during evaluation were categorized into four groups, summarized in Table IX. The largest single category was retrieval misses on multi-document queries where the relevant passage was split across a chunk boundary, followed by transcription errors on domain-specific terminology not well represented in the speech model's training data, occasional premature end-of-turn detection on pauses longer than the configured threshold, and a small number of interruption races where a barge-in arrived within a few tens of milliseconds of the response completing naturally.

TABLE IX. ERROR ANALYSIS ACROSS EVALUATION SCENARIOS

Error Category	Typical Cause	Affected Scenario Group	Rate
Retrieval miss	Relevant content split across chunk boundary	Multi-document QA	8.2%
Transcription error	Domain-specific or low-frequency terms	Single/multi-document QA	6.4%
Premature turn end	Pause exceeding threshold mid-sentence	Small talk, personal fact	3.1%
Interruption race	Barge-in within ~50 ms of natural completion	Interruption handling	4.5%

VIII. RESULTS AND DISCUSSION

The results in Tables IV through IX support the central design claim of this paper: that routing each conversational turn through only the stages it actually needs, rather than running every stage on every turn, produces a measurable latency benefit without

requiring a faster underlying language model or recognizer. The reduction in end-to-end latency relative to the sequential baseline ranges from roughly 28 percent on the most retrieval-intensive scenario to nearly 47 percent on small talk, where the proposed system skips vector search entirely. This gap narrows as the complexity of the query increases, which is expected, since a query that genuinely requires reranked retrieval over multiple documents cannot avoid paying for that work; the benefit of the orchestrator in that case comes mainly from overlapping the retrieval and generation stages rather than from skipping any stage outright.

The word error rate of the streaming transcription stage, between roughly five and seven percent across scenario groups, is consistent with figures reported for adapted streaming variants of Whisper-family models in recent work [2]-[5], and the modest increase observed on document-grounded scenarios is attributable to domain terminology drawn from the ingested PDFs that the base speech model was not specifically tuned on. This suggests that accuracy on knowledge-intensive conversations could be improved further by biasing the recognizer's vocabulary toward terms present in a session's ingested documents, which the current implementation does not yet attempt.

Memory usage scales in a way that is favorable for long-running deployments: the per-session footprint of the active buffer and summary is small and bounded, while the cost that grows with usage is concentrated in the vector index, which scales with the volume of ingested documents rather than with conversation length. This separation matters in practice because a deployment with many short-lived sessions and a small number of frequently referenced documents behaves very differently, from a resource standpoint, than one with a small number of very long sessions, and the hierarchical memory design accommodates both without requiring different configuration.

The error analysis points to retrieval misses at chunk boundaries as the most significant source of incorrect or incomplete answers, which is a known limitation of fixed-length chunking strategies in retrieval-augmented generation rather than something specific to this system's orchestration logic. Overlapping chunk boundaries during ingestion, currently set to a modest overlap fraction, partially mitigates this but does not eliminate it for passages that span more than two chunks. Interruption races, where a barge-in arrives almost simultaneously with the natural end of a response, were infrequent but did occasionally cause a brief stutter in playback; resolving this would likely require a short debounce window after a response is judged complete, during which a new speech signal is treated as the start of the next turn rather than as an interruption of the just-finished one.

From a deployment standpoint, the architecture's reliance on a single FastAPI process with an asynchronous event loop is adequate for the session volumes tested here but would need to be extended with multiple worker processes and a shared session store, rather than the in-process SQLite file used in this prototype, before serving a larger number of concurrent users. The orchestrator and memory logic are not tied to any particular process model, so this extension is expected to be a deployment change rather than an architectural one. Overall, the evaluation suggests that the four design elements introduced in Section III, hierarchical memory, context-aware retrieval, budgeted streaming, and interruption handling, work together to keep the system responsive across a range of conversational and knowledge-grounded scenarios without requiring compromises on any single capability.

IX. FUTURE WORK

Several directions follow naturally from the limitations identified during evaluation. Reducing the system's dependence on a remote language model API would benefit latency-sensitive and privacy-sensitive deployments alike, and recent progress on compact on-device language models suggests that a smaller model running locally could handle a meaningful fraction of conversational turns, falling back to a larger remote model only for queries the local model is not confident about [29]-[31]. Extending the speech pipeline to support additional languages within the same streaming architecture would broaden the system's applicability beyond the English-language evaluation reported here, building on recent multilingual speech-language modeling work [33]-[35]. The chunk-boundary retrieval misses observed in the error analysis motivate exploring overlap-aware or semantic chunking strategies, as well as prompt or context compression techniques that could allow more retrieved passages to be included within the same

token budget [25]-[27]. For deployments serving many users with similar uploaded documents, a federated or shared memory layer that allows common knowledge to be indexed once while keeping conversation history strictly private to each session would reduce redundant ingestion cost without compromising the session isolation the current design relies on. Finally, the emotion-aware response selection capability outlined in the original design goals was not implemented in the evaluated prototype, and incorporating affect-sensitive response generation, informed by recent work on empathetic dialogue with large language models, is left as a direct extension of the orchestrator's turn-planning logic [28],[29],[32].

X. CONCLUSION

This paper described a real-time conversational voice agent organized around an Adaptive Conversation Orchestrator rather than a fixed chain of speech recognition, language generation, and speech synthesis. By coordinating a hierarchical memory structure, an intent-aware retrieval module, a shared latency budget across pipeline stages, and an interruption handling mechanism, the system maintains the contextual continuity expected of a long-running conversation while keeping response latency within a range suitable for live spoken interaction. The implemented prototype, built with a FastAPI backend, a Next.js frontend, Whisper-based streaming transcription, and ChromaDB-backed retrieval, was evaluated across small talk, personal-fact recall, and single- and multi-document question answering, and the results indicate that routing each turn through only the pipeline stages it requires yields a substantial latency reduction relative to a conventional sequential pipeline, with the reduction being largest for conversational turns and smallest for the most retrieval-intensive queries, as expected. The error analysis and resource measurements point to concrete, addressable limitations, chiefly chunk-boundary retrieval misses and the single-process deployment model, rather than to any fundamental shortcoming in the orchestration approach itself, suggesting that the architecture is a practical foundation for further work on multilingual support, on-device inference, and affect-aware response generation in conversational voice agents.

REFERENCES

[1] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," in Proc. Int. Conf. Mach. Learn. (ICML), 2023, pp. 28492-28518.

[2] D. Macháček, R. Dabre, and O. Bojar, "Turning Whisper into a real-time transcription system," arXiv:2307.14743, 2023.

[3] Anonymous, "CarelessWhisper: Turning Whisper into a causal streaming model," arXiv:2508.12301, 2025.

[4] Anonymous, "WhisperPipe: A resource-efficient streaming architecture for real-time automatic speech recognition," arXiv:2604.25611, 2026.

[5] Anonymous, "Pushing the limits of on-device streaming ASR: A compact, high-accuracy English model for low-latency inference," arXiv:2604.14493, 2026.

[6] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS), 2020, pp. 9459-9474.

[7] Anonymous, "A comprehensive survey of retrieval-augmented generation (RAG): Evolution, current landscape and future directions," arXiv:2410.12837, 2024.

[8] Anonymous, "A systematic review of key retrieval-augmented generation (RAG) systems: Progress, gaps, and future directions," arXiv:2507.18910, 2025.

[9] Anonymous, "Retrieval-augmented generation: A comprehensive survey of architectures, enhancements, and robustness frontiers," arXiv:2506.00054, 2025.

[10] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP), 2019, pp. 3982-3992.

[11] N. Muennighoff, N. Tazi, L. Magne, and N. Reimers, "MTEB: Massive text embedding benchmark," in Proc. Annu. Meeting Assoc. Comput. Linguistics (ACL), 2023.

[12] Anonymous, "Repetition improves language model embeddings," arXiv:2402.15449, 2024.

[13] J. Wu et al., "DuplexMamba: Enhancing real-time speech conversations with duplex and streaming capabilities," arXiv:2502.11123, 2025.
[14] Anonymous, "PredGen: Accelerated inference of large language models through input-time speculation for real-time speech interaction," arXiv:2506.15556, 2025.
[15] Anonymous, "From static inference to dynamic interaction: A survey of streaming large language models," arXiv:2603.04592, 2026.
[16] Anonymous, "From turn-taking to synchronous dialogue: A survey of full-duplex spoken language models," arXiv:2509.14515, 2025.
[17] Anonymous, "LLM-enhanced dialogue management for full-duplex spoken dialogue systems," arXiv:2502.14145, 2025.
[18] Anonymous, "Semantic-aware interruption detection in spoken dialogue systems: Benchmark, metric, and model," arXiv:2603.24144, 2026.
[19] Anonymous, "Speculative end-turn detector for efficient speech chatbot assistant," arXiv:2503.23439, 2025.
[20] Anonymous, "Thai semantic end-of-turn detection for real-time voice agents," arXiv:2510.04016, 2025.
[21] Q. Wang et al., "Recursively summarizing enables long-term dialogue memory in large language models," arXiv:2308.15022, 2023.
[22] Anonymous, "A survey of context engineering for large language models," arXiv:2507.13334, 2025.
[23] Anonymous, "On memory construction and retrieval for personalized conversational agents," arXiv:2502.05589, 2025.
[24] Anonymous, "Reflective memory management for long-term conversational agents," in Proc. Annu. Meeting Assoc. Comput. Linguistics (ACL), 2025.
[25] Anonymous, "Prompt compression for large language models: A survey," arXiv:2410.12388, 2024.
[26] Anonymous, "Prompt compression with context-aware sentence encoding for fast and improved LLM inference," arXiv:2409.01227, 2024.
[27] J. Mu, X. L. Li, and N. Goodman, "Learning to compress prompts with gist tokens," in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS), 2023.
[28] Anonymous, "Emotion Omni: Enabling empathetic speech response generation through large language models," arXiv:2508.18655, 2025.
[29] Anonymous, "Enhancing emotional generation capability of large language models via emotional chain-of-thought," arXiv:2401.06836, 2024.
[30] Anonymous, "A survey of small language models," arXiv:2410.20011, 2024.
[31] Anonymous, "Collaborative inference and learning between edge SLMs and cloud LLMs: A survey of algorithms, execution, and open challenges," arXiv:2507.16731, 2025.
[32] Anonymous, "Local-cloud inference offloading for LLMs in multi-modal, multi-task, multi-dialogue settings," arXiv:2502.11007, 2025.
[33] M. Nguyen et al., "Ichigo: Mixed-modal early-fusion realtime voice assistant," arXiv:2410.15316, 2024.
[34] N. Das et al., "SpeechVerse: A large-scale generalizable audio language model," arXiv:2405.08295, 2024.
[35] Anonymous, "Summary on the multilingual conversational speech language model challenge: Datasets, tasks, baselines, and methods," arXiv:2509.13785, 2025.
[36] OpenAI, "GPT-4 technical report," arXiv:2303.08774, 2023.