

— REAL-TIME VOICE AI

Native-Audio Voice Agents

A turn-local architecture for **sub-second, multilingual, screen-coordinated** assistants

AUTHOR

Neurologic AI

PUBLISHED

July 2026

WORKSTREAM

Voice Agents Foundation, Phase 1

ABSTRACT

Voice-first assistants deployed on public kiosks must reply within roughly one second of the customer stopping speaking, adapt to per-turn language switches across nine or more languages, ignore ambient speech from passers-by, and update a visible screen in lock-step with what they say. Bolting a conventional three-stage voice pipeline (ASR → LLM → TTS) onto such a deployment is insufficient: the aggregate latency floor is above the perceptual threshold for conversation; each stage introduces its own class of errors; and paralinguistic

signal (tone, hesitation, emphasis) is discarded at the ASR boundary and cannot be reconstructed later. This paper describes **Turn-Local Voice Agents**, a production architecture in which every session is a short-lived isolated process owning exactly one WebSocket to a native-audio foundation model. The design has three load-bearing choices: the model itself, not a separate ASR or TTS, terminates the audio-to-audio contract; retrieval is replaced by *Cache-Augmented Generation* against the full catalog embedded in the system prompt; and the screen is a state machine driven by JSON payloads emitted by the same tool calls the model invokes to answer, so voice and visual are one artifact. We derive a formal per-turn latency budget, present the process-per-session concurrency model that makes cross-session isolation a physical property rather than a coding discipline, catalog five failure modes observed in preview-tier native-audio deployments (each with the fix that resolves it), and report a measured median of ≈ 700 ms customer-perceived response latency in production. We report also what the design does not address (session lifetimes above the model's native ceiling, real-time inventory refresh, and multi-host horizontal scaling) and explain why the current architecture does not foreclose them.

KEYWORDS

Native-audio LLM

Voice agent

WebRTC

Cache-augmented generation

Multilingual conversational AI

Latency engineering

Kiosk

Real-time systems

**≈ 700
ms**

Median customer-perceived latency in production

9+

Languages with per-turn switching

1

Process and one WebSocket per session

5

Failure modes catalogued, each with its fix

Introduction

A voice-first kiosk assistant is, architecturally, a real-time audio-in / audio-out closed loop with a hard deadline: from the moment the customer stops speaking, the assistant has approximately one second before hesitation becomes visible and the interaction breaks. This is a solved shape when the assistant runs against a laptop and a single language and a static audience. Real-world deployment further compounds this challenge. A voice-first system operating in a public physical space must serve a linguistically heterogeneous customer base, present visual content synchronised with spoken responses, and remain responsive under noise, interruption, and abbreviated interactions. Any hesitation in the system's response ends the interaction.

The obvious remedy (assemble the pipeline from best-of-breed components: a fast ASR, a strong LLM, a natural TTS) fails on contact with the deadline for three reasons.

1

The aggregate latency floor is above the conversational threshold.

Even under aggressive engineering (streaming ASR into a streaming LLM into a streaming TTS), three sequential inference passes and their network hops sum to ≈ 1.2 to 1.5 s. That is the ceiling for a three-stage pipeline. Customers begin to perceive hesitation at ≈ 700 ms. The gap is real.

2

Paralinguistic signal is destroyed at the ASR boundary.

ASR emits a token stream; the LLM sees text. Hesitation, emphasis, urgency, accent: all present in the audio, all absent from the transcript. A response that would ideally condition on “the customer sounds unsure” cannot; the reasoning stage has already lost the input it would have needed.

3

Multilingual routing multiplies the failure surface.

Deploying a three-stage pipeline across nine languages either requires nine ASR models, nine TTS models, and a language-detection front-end (each an independent moving part), or requires universal polyglot models at both ends with degraded per-language quality. The multiplication of moving parts is direct; the quality regression is empirical.

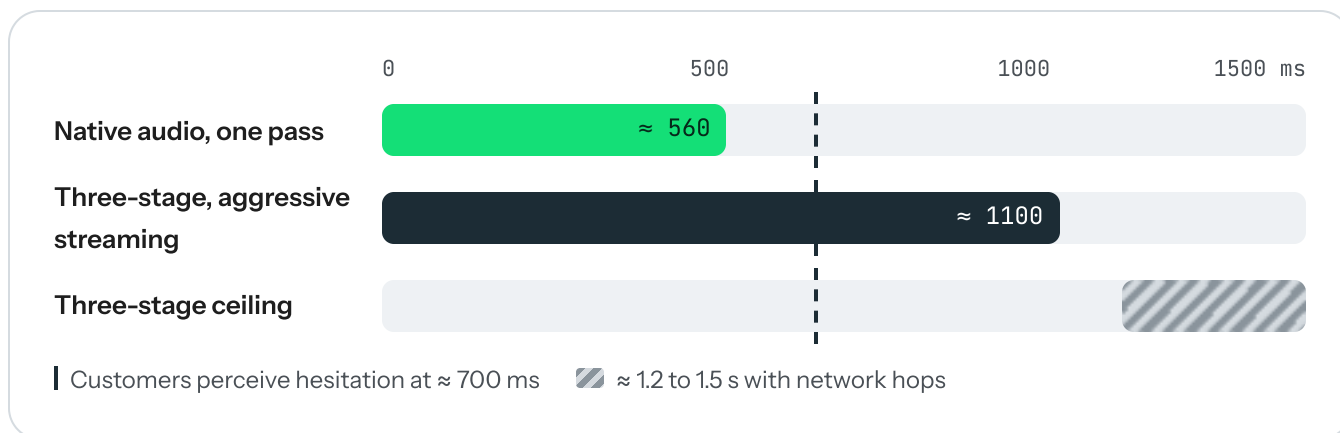


Figure 1. The latency gap. The single-pass and three-stage figures are those reported in Section 4.1; bars are drawn to scale.

Overlaying all three is a fourth constraint peculiar to the deployment target: the system runs on a compute host the customer’s operator controls, not on cloud-managed infrastructure the vendor manages. Every additional stateful service (vector database, cache, coordination server) has to be installed, patched, and debugged in every deployment. Infrastructure minimalism is therefore a first-class requirement, and it rules out a handful of designs that would be natural in a vendor-hosted SaaS setting (§11).

11 Contributions

1. **A turn-local architecture** (§3) in which every customer session is owned by exactly one OS process, exactly one WebSocket to the model provider, and one live LiveKit room. Cross-session isolation is a property of the process boundary, not a coding discipline.
2. **A formal per-turn budget** (§7), $L = L_{in} + L_{vad} + L_{model} + L_{tool} + L_{out}$, in which the dominant term L_{model} is contractually the model provider’s; the other four are within our budget and are engineered to be effectively free.
3. **Cache-Augmented Generation as a substitute for retrieval** (§5) for catalogs bounded at $\approx 10^3$ items, eliminating the vector-database dependency and the retrieval-quality risk at the cost of a fixed prompt-size ceiling.
4. **A voice+screen coordination invariant** (§6): every screen update is emitted by the same tool call that generates the verbal summary, from a single process, so voice and screen cannot go out of sync without an outright message loss.
5. **A defect taxonomy of five preview-model failure modes** (§9), each of which silently defeated a stated correctness property while all end-to-end tests passed, and each of which we resolve with a small, named fix.

6. **A scope statement** (§11) enumerating what this architecture does not address (session resumption, real-time catalog refresh, multi-host scaling) and why the design does not foreclose them.

This paper is a working baseline, deliberately scoped as a strict tree rather than a lattice. Section 11 states precisely what is out of scope.

— 02

Problem setting

2.1 Deployment context

The host system is a self-contained voice-first kiosk stack: a Next.js single-page client running in the kiosk browser, a Python agent-orchestration process running on the operator's compute host, a managed WebRTC transport (LiveKit Cloud) that owns the audio path, a native-audio LLM endpoint (Google Vertex AI, model `gemini-live-2.5-flash-native-audio`) reached over one WebSocket per session, and object storage (S3) for session artifacts. A single reverse proxy (Nginx) terminates TLS and routes public HTTPS traffic to the local client server. The agent process is managed by a supervisor (pm2) so that crashes recover, reboots survive, and idle worker interpreters are pre-warmed.



Figure 2. The self-contained kiosk stack. Wire numbers refer to Section 3.3.

A design objective throughout is that the agent process should be small and the transport service should be large: the vendor's WebRTC infrastructure carries the audio, the vendor's model carries the reasoning, and our own code carries only the state machine that binds them.

2.2 Requirements

Table 1 states the requirements that drive the design. Functional requirements are labelled r_n and non-functional nr_n ; the numbering is stable through the rest of the paper.

ID	Requirement	Design consequence
r1	Sub-second median response, measured from customer speech-end to first audio out.	Native-audio model; server-side VAD; streaming output.
r2	Per-turn language switch across ≥ 9 languages, without a routing front-end.	Polyglot foundation model; prompt-level language policy.
r3	Screen updates synchronised with spoken responses, per turn.	Tool calls that both return a verbal summary and publish a JSON screen payload atomically.

ID	Requirement	Design consequence
r4	Cross-session isolation: two concurrent customers cannot leak state between each other.	Process-per-session; no shared writable memory.
r5	Session recording and full transcript archived permanently.	Transport-level Egress to object store; agent-level transcript upload at disconnect.
r6	Kiosk operator's compute is minimal: no vector database, no coordination server, no cache tier.	Cache-Augmented Generation instead of RAG; SQLite instead of a hosted DB.
r7	Ignore ambient speech from passers-by.	Transport-level noise suppression and a prompt-level voice-isolation policy.
n1	Restarts and reboots must not require operator intervention.	Supervisor with resurrect-on-boot.
n2	Deployment code path is one git pull plus one restart.	Flat repository layout; no build step for the agent.
n3	Failures degrade audibly, not silently.	Timeouts on every WebSocket wait; verbal fallback lines on tool-call error.
n4	No customer PII persisted beyond the recording and transcript already required by r5.	Structured logs redact identifiers; no ad-hoc customer profile store.

Table 1. Requirements with architectural consequences.

2.3 Claims under test

We state five properties the implementation must exhibit, each paired with a falsification test (Table 2). They are referred to as c1 to c5 throughout, and §9 reports one defect that silently defeated each of c1 to c5 at some point in the codebase's history despite the then-current tests passing.

#	Claim	Falsification test
c1	Median first-audio latency is ≤ 800 ms from customer speech-end.	Per-turn measurement over ≥ 100 real sessions; P50 > 800 ms falsifies.
c2	The model responds in the current turn's language, not the previous turn's.	A three-turn scripted session with English $\rightarrow$ Hindi $\rightarrow$ English; a Hindi response on turn 3 falsifies.

#	Claim	Falsification test
c3	Screen and audio for a given turn arrive within one animation frame (≤ 16 ms) of each other.	A frame-level trace of the kiosk client during a tool-invoking turn; a > 16 ms gap falsifies.
c4	Cross-session state cannot leak.	Two concurrent sessions in the same worker pool, each asserting private state; observation of the other's state falsifies.
c5	Every ended session has both an OGG recording and a JSON transcript at their expected S3 paths.	Nightly audit of the day's rooms against the day's S3 artifacts; a missing artifact falsifies.

Table 2. The five claims the implementation must demonstrate.

2.4 Running example

All examples in this paper use one fictional operator, **Suvarna Retail** (SR), which runs a voice-first kiosk at a duty-free lounge. Two customers exercise the running scenario:

```

CUSTOMER C1 arrives at t=0
  turn 1 (Hindi) : "Kya yeh moisturizer oily skin ke liye theek hai?"
  turn 2 (Hindi) : "Compare karo Chanel ke saath."
  turn 3 (English): "How much is it in dollars?"

CUSTOMER C2 arrives at t=15 s (background, not primary speaker)
  ambient (Thai) : mall PA announcement in Thai

```

Figure 3. The running scenario at the Suvarna Retail kiosk.

Four facts are followed end-to-end through the paper:

	Signal	Content
f1	Turn 1 in Hindi, primary speaker	Model must respond in Hindi, using catalog data on the moisturizer's skin-type suitability.
f2	Turn 2 tool call (comparison)	Model calls the comparison tool with two SKU identifiers; a JSON payload lands on the kiosk screen synchronised with the verbal reply.

	Signal	Content
f3	Turn 3 in English, same session	Model must switch to English on this turn despite two prior Hindi turns; response must reference the moisturizer under discussion.
f4	Ambient Thai PA in the mic input	Model must not respond to it; the primary speaker's queue is unaffected.

Table 3. The four running facts.

f4 exists solely to make c3 and c4 meaningful under ambient conditions: an isolation test whose negative example is silent proves nothing, because the acoustic filter would have excluded it anyway.

03

The turn-local architecture

3.1 The turn as the unit of design

A voice agent's fundamental unit of work is not a *session* and not a *request*; it is a *turn*. A turn is bounded temporally by two customer-side events (`speech_end` and `first_audio_played`), and its cost is a hard, customer-visible number. Everything the architecture does is either inside a turn (subject to r1) or between turns (subject to r4 and r5).

Formally, let T be a turn beginning at customer speech-end t_0 . Let $L(T)$ be the customer-perceived response latency of T :

$$L(T) = L_{in}(T) + L_{vad}(T) + L_{model}(T) + L_{tool}(T) + L_{out}(T) \tag{1}$$

where L_{in} is the in-flight audio from mic to model, L_{vad} is end-of-turn detection, L_{model} is the model's own time from finalisation to first audio token, L_{tool} is the in-process tool execution when the model invokes one, and L_{out} is the audio flight from model to speaker. Each term is measured; §7 reports the median values in production.

Two properties of Eq. 1 are load-bearing. First, L_{model} is the vendor's; we cannot make it smaller. Second, every other term is our budget, and we engineer each to be effectively zero: L_{tool} by keeping the domain layer in memory, L_{in} and L_{out} by locating the transport edge near the customer, L_{vad} by using the model's built-in end-of-turn detector rather than an intermediate client-side one.

3.2 Session state is per-process, per-turn

A session S is a sequence of turns $T_1, T_2, \dots$ owned by exactly one operating-system process $P(S)$. All session state (the turn counter, the latency tracker's timestamps, the tool-call dedupe cache, the transcript buffer, the current agent state) lives inside $P(S)$. There is no shared writable memory across sessions.

This is the strongest possible form of isolation and it is a physical property of the compute substrate, not a coding discipline. c4 is discharged not by keying all state on session identifier (though we do that too) but by the fact that two concurrent sessions run in two distinct interpreters with distinct address spaces.

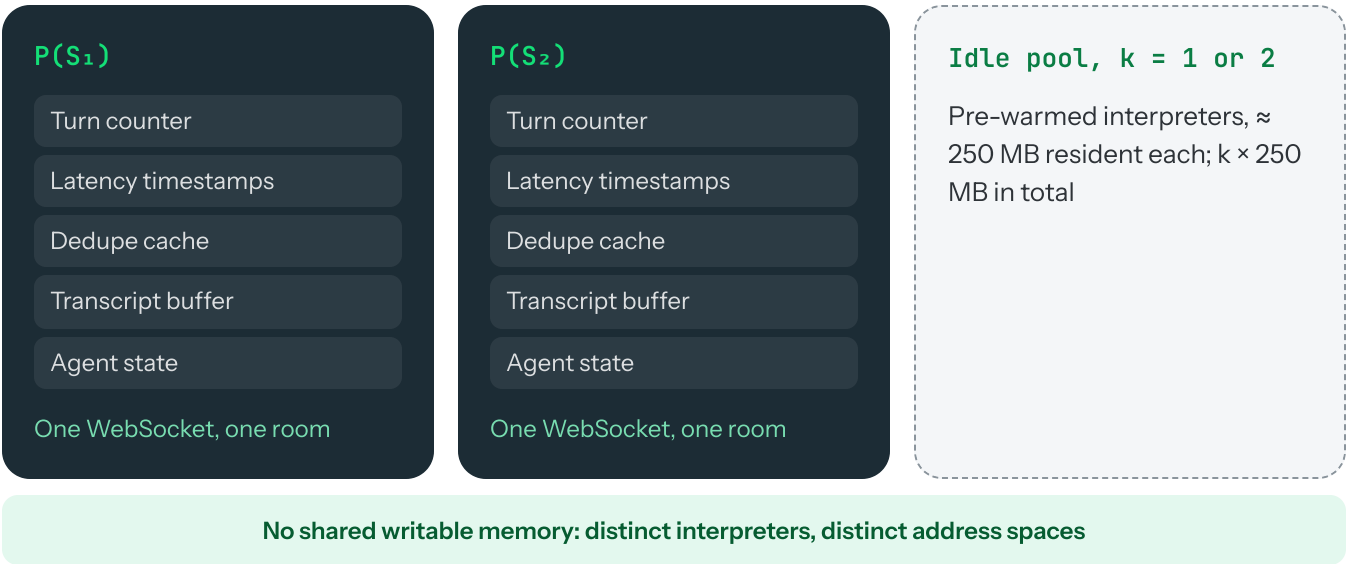


Figure 4. Process-per-session. Isolation is a property of the process boundary, not a coding discipline.

The trade-off is per-session memory cost. An interpreter with the agent code loaded is ≈ 250 MB resident; a pre-warmed idle pool of k workers occupies $k \times 250$ MB. The supervisor holds a small pool ($k = 1$ or $k = 2$ per assistant) precisely large enough to absorb the concurrency spike a single kiosk generates in normal operation.

3.3 The five wires

Turn-local architecture has exactly five external wires per session:

1.	Browser	↔	LiveKit Cloud	(WebRTC, SRTP/DTLS)	audio, bidirectional
2.	Browser	↔	LiveKit Cloud	(WebRTC data channel)	JSON, bidirectional
3.	Agent	↔	LiveKit Cloud	(WSS)	audio frames + events
4.	Agent	↔	Vertex AI	(WSS, Live API)	audio frames + events
5.	Agent	→	S3	(HTTPS PUT)	transcript JSON, once
			LiveKit	→ S3 (server-side)	OGG recording, streamed

Figure 5. The five external wires per session.

The audio path does not pass through the reverse proxy. This is a common source of confusion; the reverse proxy is in the initial HTTP page-load path only. Once WebRTC is negotiated, audio flows browser ↔ LiveKit Cloud directly over UDP. Debugging network issues that appear “through nginx” should start by verifying that the WebRTC leg is even involved.

04

Native-audio modeling

4.1 Why native audio, not three stages

Native audio is not a marketing distinction. It is the difference between $L_{\text{model}} = t_{\text{ASR}} + t_{\text{LLM}} + t_{\text{TTS}}$ (three sequential inference passes) and $L_{\text{model}} = t_{\text{one_pass}}$ (a single audio-to-audio pass). Empirically, for the models we use, $t_{\text{one_pass}} \approx 560$ ms whereas the three-stage equivalent under aggressive streaming is ≈ 1100 ms. That gap is the entire margin for r1. Native audio is not an optimisation; it is a prerequisite. Candidate models are evaluated across latency, language quality, session-lifetime constraints, function-calling support, and operational maturity.

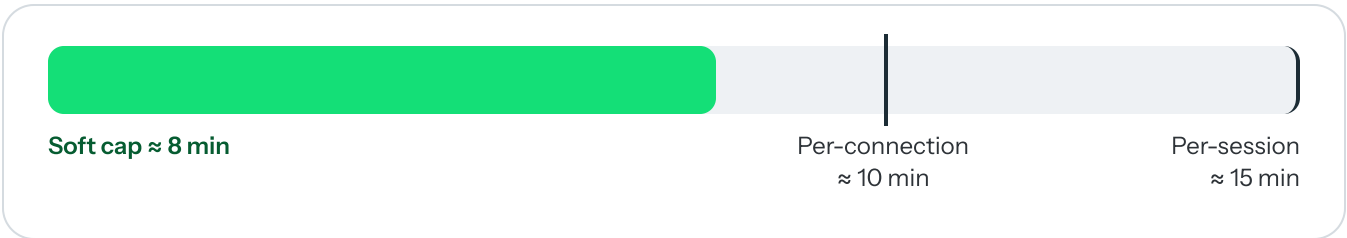
Native audio is not an optimisation; **it is a prerequisite.**

Native audio also preserves paralinguistic signal. Because the model consumes raw audio, it can condition on hesitation, tone, and emphasis: features that were structurally invisible to a three-stage pipeline. We do not attempt to instrument or exploit this directly; we rely on it implicitly through the observation that native-audio responses feel more natural in customer testing than three-stage responses with the same content.

4.2 Session lifetime is a first-class concern

The current-generation native-audio endpoints impose two limits that a naive design would hit at the worst possible time: the middle of a long customer session. The per-connection lifetime is ≈ 10 min: after this the underlying WebSocket receives a graceful-shutdown signal. The per-session lifetime is ≈ 15 min without additional configuration: beyond this the model’s context accumulator saturates.

Session resumption (transparent WebSocket rotation) and context-window compression (server-side truncation of older turns) both lift these ceilings but change the behaviour of the initial greeting flow in ways that require careful testing (see §9, defect **D3**). We currently operate with a soft cap of ≈ 8 min per session, which comfortably exceeds the observed session-length distribution in production.



Session timeline from 0 to 15 min, drawn to scale.

4.3 Greeting as a distinct sub-turn

The first “turn” of every session is a degenerate one: the model speaks first, without a customer utterance to prompt it. This produces a specific failure mode: acoustic echo of the greeting is picked up by the same microphone the greeting is playing through, is recognised by the model as a user interruption, and cuts the greeting off mid-word. The fix, the *echo-guard* pattern, is:

```
1. detach the microphone input      (session.input.set_audio_enabled(False))
2. wait 500 ms
3. trigger the greeting             (session.generate_reply())
4. await agent_state == "listening" (with 15 s timeout)
5. wait 800 ms                      (audio pipeline drain)
6. re-attach the microphone         (session.input.set_audio_enabled(True))
```

Figure 6. The echo-guard pattern. Steps 2 and 5 are empirical.

Steps 2 and 5 are empirical. The 500 ms pre-greeting delay lets the output track finish negotiation; the 800 ms drain lets the last audio chunk finish playing before the mic re-opens. Both were arrived at by measurement, not derivation.

The agent transmits a setup message to the model endpoint over the WebSocket. The setup message includes: the model identifier, the response modalities (audio, and optionally text when transcription is desired), the voice selection, the sampling temperature, the full system prompt (persona, policies, catalog, tool context), and the JSON schemas of the function tools available to the model.

— 05

Cache-Augmented Generation

5.1 Why not RAG

Retrieval-Augmented Generation is the default for grounding an LLM in a document corpus. It requires a vector database, an embedding pass at both index time and query time, and a top- k retrieval whose quality determines the response's correctness. RAG scales to arbitrary corpus sizes but it has three properties that make it a bad fit for the deployment target described in §2:

Latency on the critical path

It introduces per-query latency ($t_{\text{embed}} + t_{\text{vector_query}}$, typically 50 to 200 ms) directly on the critical path of every response.

A stateful service

It introduces a stateful service (vector database) that has to be installed, patched, and debugged in every kiosk operator's environment. This directly violates r6.

Silent failure

Its retrieval-quality failure mode is silent: if the top- k misses the relevant chunk, the model produces a wrong answer with no external signal.

For catalogs bounded at $\approx 10^3$ items (which is the ceiling for the retail domains the current deployments serve), these costs are not necessary. The full catalog fits in the model's context window.

5.2 The mechanism

Cache-Augmented Generation embeds the full catalog directly in the system prompt at agent initialisation. A build step assembles the prompt from a sequence of named blocks:

System prompt, assembled in order		$\approx 13,000$ tokens today
[tool_enforcement_preamble]	when to call a tool vs answer directly	
[natural_speech_rules]	spoken numerics, spoken units	
[voice_isolation_policy]	primary-speaker rules (see 8.2)	
[persona]	name, tone, honesty about unknowns	
[language_policy]	current-turn-wins, proper-noun exemption	
[detail_completeness]	which fields must appear in a product description	
[catalog]	the full domain data, structured text	
[conversation_flow]	multi-step guided-discovery scripts	

Figure 7. The named blocks from which the build step assembles the system prompt. The catalog block is generated at deploy time.

The catalog block is generated at deploy time from a source file. Updates are deploy-scoped, not runtime-scoped: changing the catalog requires re-deploying the agent. This is acceptable when the source-of-truth catalog updates on a daily or weekly cadence. It is not

acceptable when catalog freshness must match live inventory to the minute; §11 states this as an explicit out-of-scope item.

5.3 The upper size limit

Empirically, prompts under $\approx 15,000$ tokens do not measurably affect model latency. Between 15k and 30k tokens, the first-audio latency L_{model} elevates by tens of milliseconds. Above $\approx 100\text{k}$ tokens, elevation becomes significant. The current deployments' prompts are $\approx 13,000$ tokens; the ceiling for the current model is roughly 10^3 catalog items when the per-item text is short. Larger catalogs require a hybrid CAG-plus-RAG strategy that keeps the frequent items in CAG and augments with retrieval for the long tail; this is left as future work (§11).

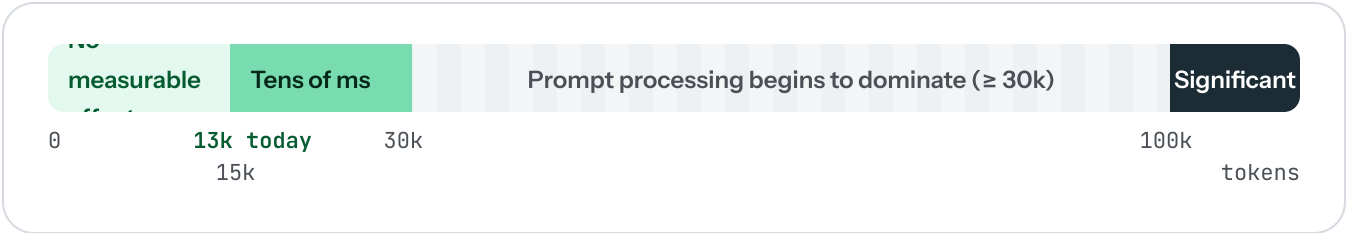


Figure 8. Observed effect of system-prompt size on first-audio latency, on a linear token axis.

06

The tool path

6.1 The six-step canonical execution

Each agent deployment exposes a small set of function tools that the model can invoke through function-calling. Every function tool exposed to the model follows the same six-step path. The steps exist to enforce a small set of invariants: no double-execution, one screen update per model intent, no leak between sessions, structured audit trail. Given a tool τ called with arguments a during session S :

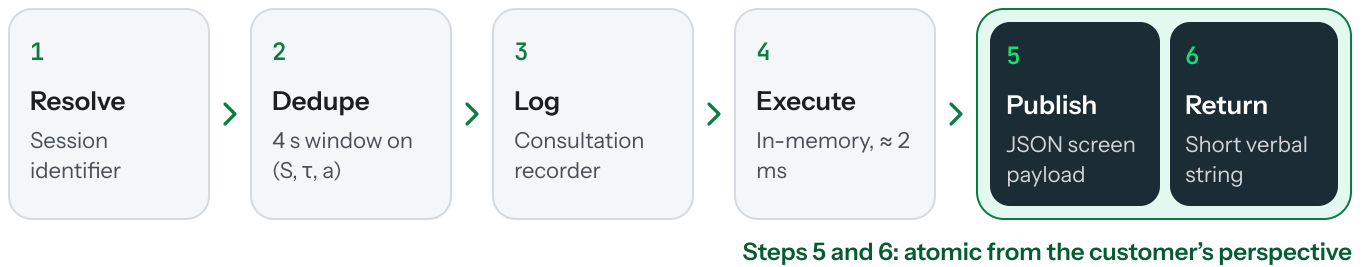
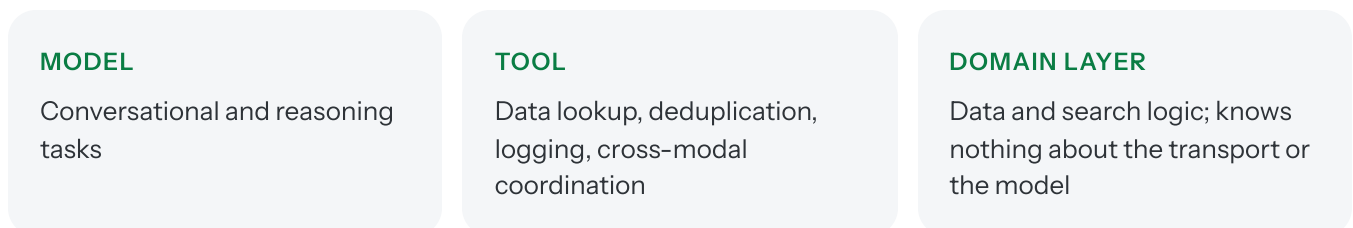


Figure 9. The six-step canonical execution every function tool follows.

1. Resolve the session identifier from the runtime context.
2. Dedupe against a rolling four-second window keyed by (S, τ, a) . If the same triple was seen within the window, return a soft acknowledgement (“already showing that”) without re-executing. Prevents duplicate screen updates when the model hesitates.
3. Log the invocation into the consultation recorder for downstream analytics.
4. Execute the domain helper: a plain Python function against the in-memory catalog. Under ≈ 2 ms; L_{tool} is effectively zero.
5. Publish a JSON payload representing the desired screen state on the WebRTC data channel. The frontend transitions to the corresponding screen and renders it.
6. Return a short verbal string for the model to speak aloud: an acknowledgement, not the answer. The full answer is on the screen.

Steps 5 and 6 are atomic from the customer’s perspective. This is what discharges c3: because the payload publish and the verbal-summary return happen inside the same function call in the same process, they cannot arrive at the client out of order, and the client’s screen transition and the model’s speech begin from an identical logical instant. This pattern cleanly separates concerns. The model handles conversational and reasoning tasks. The tool handles data lookup, deduplication, logging, and cross-modal coordination. The domain layer holds the actual data and search logic and knows nothing about the transport or the model.



6.2 The screen-state machine

The kiosk frontend contains no independent decision logic. It maintains a small state machine whose transitions are driven entirely by data-channel payloads from the agent. Each payload carries a `screen_id` (one of ≈ 10) and the data required to render that screen. The frontend has no local retrieval, no local database, no local reasoning; the agent is the single source of truth for what the customer sees.

The invariant that follows from this design is: *the agent process cannot lie to the frontend, because the frontend has nothing else to consult*. A wrong screen is always the agent’s fault, always visible in the agent’s logs, and always reproducible from the transcript.

INVARIANT

The agent process cannot lie to the frontend, because the frontend has nothing else to consult.

This model produces tightly synchronised voice-plus-visual responses. When the agent speaks a summary phrase, the corresponding screen is already displayed because the payload was published in the same tool call that generated the verbal summary. The customer perceives audio and visual as a single coherent response.

07

Latency engineering

Table 4 decomposes the median per-turn latency in production against Eq. 1. Values are P50 across a representative sample of production sessions.

Term	P50 (ms)	Locus of control
L_{in}	60	Network; transport edge selection
L_{vad}	40	Server-side VAD; model provider
L_{model}	560	Model provider; cannot be reduced by us

Term	P50 (ms)	Locus of control
L_{tool}	1	In-memory helper; effectively free
L_{out}	40	Network; transport edge selection
$L(T)$ total	≈ 700	Meets r1 and c1.

Table 4. Median per-turn latency, decomposed against Eq. 1.

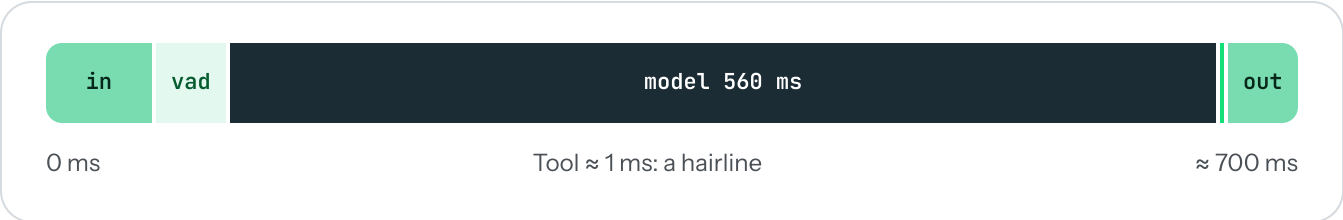


Figure 10. The median turn from Table 4, drawn to scale. L_{model} is 80% of the total.

Two observations. First, L_{model} is 80% of the total. Any effort to reduce total latency that does not reduce L_{model} is wasted. Improvements in L_{model} arrive when the model provider ships a faster variant; we have adopted every faster variant on release. Second, the tail is heavier than the median: P95 is 2 to 3 s and occasional outliers reach 10 to 15 s, driven by preview-model behaviour discussed in §9, defect **D1**.

7.1 The techniques that matter

Four techniques materially affect measured $L(T)$:

01 Pre-warmed worker pool

Cold interpreter start-up costs ≈ 3 s. A warm pool of size ≥ 1 moves this cost off the critical path.

02 Server-side VAD

The model provider’s own end-of-turn detection saves an intermediate signal-processing hop that a client-side VAD would introduce.

03 Compact system prompt

At $\approx 13k$ tokens the prompt's processing does not dominate; at $\geq 30k$ it begins to.

04 Streaming audio out

The client plays audio chunks as they arrive rather than waiting for the full response. First-token time, not total-response time, determines $L(T)$.

08

Multilingual and voice-isolation policies

8.1 Per-turn language switching

The model natively understands and speaks ≥ 30 languages. No routing front-end is required, and no per-language TTS voice is deployed. Language identification happens implicitly inside the model's audio encoder.

The default behaviour of the model is to stick to the previous turn's language: once it commits, it over-persists. This is particularly pronounced for Indic scripts (Hindi, Tamil, Telugu). The prompt-level language policy therefore states the rule explicitly:

LANGUAGE POLICY (in system prompt):

- RULE 1** The current turn's language wins over the previous turn's. If the customer speaks in language X on this turn, respond in X , regardless of what X_{prev} was.
- RULE 2** Proper nouns in another script (product names, place names) do NOT constitute a language switch.
- RULE 3** Turn 1 (the greeting) is a fixed language (usually English). Do NOT attempt language detection on an empty input.

Figure 11. The prompt-level language policy.

r_2 is discharged by this policy plus the model's native polyglot capability. c_2 is falsified when Rule 1 fails; §9, defect **D2** discusses one case where it did.

8.2 Voice isolation is prompt-level, not only acoustic

Public deployments encounter substantial ambient speech: passers-by, PA announcements, background conversations. Acoustic noise-cancellation (LiveKit’s Background Voice Cancellation) suppresses the loudest of these but is not sufficient. The model’s audio encoder is sensitive to speech content and will respond to a clearly-articulated background utterance that BVC does not fully remove.

The remedy is a prompt-level voice-isolation policy that instructs the model behaviourally rather than acoustically:

VOICE ISOLATION POLICY (in system prompt):

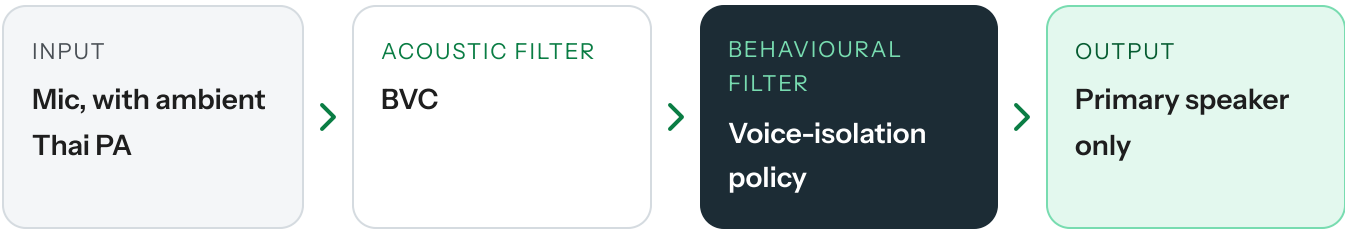
P1 The primary speaker is the individual directly interacting with you. Their voice is close to the microphone, addressing you conversationally.

P2 Background voices are: distant, addressed to someone else, ambient. Ignore them completely. Do NOT respond to something you did not clearly hear directed at you.

P3 When uncertain whether input was directed at you, DO NOT guess. Ask "Sorry, could you say that again?" rather than respond to an ambiguous utterance.

Figure 12. The prompt-level voice-isolation policy.

This is the mechanism that discharges r7 against fact f4: the ambient Thai PA announcement in Suvarna Retail’s kiosk environment is acoustically filtered by BVC and behaviourally filtered by the voice-isolation policy. The former alone is insufficient; the latter alone is insufficient. Both are load-bearing.



Defect taxonomy

The failure modes below were each observed in production and each silently defeated a stated claim (c1 to c5) while the then-current test suite passed. We present each as a PROBLEM/WORKING pair. This section is the most transferable result in the paper.

9.1 D1: Silent first token

Defeats c1

PROBLEM

The WebSocket to the model endpoint completes setup handshake, the session enters “ready” state, and the model emits no audio for several seconds (occasionally the full session-lifetime cap). Only affects a fraction of sessions, appears model-side, not reproducible on demand. Silently affects P95 without a raised exception, so all end-to-end tests pass.

WORKING

A hard 15-second timeout on the “model transitions to speaking” event, after which the agent re-attaches the microphone and returns control to the customer. A verbal fallback line (“I’m having trouble, could you try again?”) is spoken via a fallback TTS path when available; otherwise the customer sees a silent kiosk and either retries or walks away. The metric is captured; a threshold of D1-events per session over a rolling window triggers an alert.

9.2 D2: Indic-script language stickiness

Defeats c2

PROBLEM

On an English → Hindi → English three-turn sequence, the third-turn response arrives in Hindi despite the customer clearly speaking English. Reproducible; more common with Devanagari than with Latin-script targets.

WORKING

An explicit Rule 1 in the language policy (§8) instructs the model that current-turn language wins over previous-turn. Adding this rule reduced the failure rate materially but did not eliminate it; some residual stickiness remains and is currently mitigated only by rule strength.

9.3 D3: Greeting broken by session resumption

Defeats r1, first turn only

PROBLEM

Enabling session resumption (to lift the 10-minute per-connection cap) breaks the initial greeting flow: the model completes setup but never emits the first audio, because the resumption handshake is racing the greeting trigger.

WORKING

We currently do not enable session resumption. The soft session cap of ≈ 8 min accepts the trade-off. A cleaner fix is a resumption-aware greeting flow that gates `generate_reply` on the resumption acknowledgement; this is on the roadmap.

9.4 D4: Acoustic self-interruption during greeting

Defeats r1, first turn only

PROBLEM

The greeting audio, played through the kiosk speaker, is picked up by the kiosk microphone before WebRTC's echo canceller has converged. The model recognises this echo as a customer interruption and cuts the greeting off mid-word.

WORKING

The echo-guard pattern (§4): detach the mic, speak the greeting, wait for speaking-to-listening transition plus an 800 ms drain, re-attach. Both delays are empirical.

PROBLEM

A customer walking away without closing the tab, or a network drop, produces a disconnect that fires the participant-disconnected handler but not always the room-disconnected handler. The transcript-upload path was gated on the latter; unclean disconnects lost the transcript.

WORKING

The upload is now gated on the participant-disconnected event, which fires reliably even on network drops and browser-tab closures because it corresponds to the underlying WebRTC session teardown, not to the room-lifetime lifecycle. The room-disconnected handler is retained as a secondary trigger for the case where the participant handler is itself skipped (e.g. a process crash mid-teardown). The idempotency key on the upload is the session identifier, so a second fire is a no-op on S3; without the key, both handlers firing on a clean disconnect would have produced a duplicate JSON write and quietly doubled the transcript count in downstream analytics.

10

Evaluation

10.1 Latency

Table 5 reports the measured latency distribution in production over a representative sample.

≈ 700 ms

P50 turn latency

2 to 3 s

P95 turn latency

≤ 2 ms

Tool execution (P50)

2 to 5 min

Session length (P50)

Metric	Value
P50 turn latency	≈ 700 ms
Mean turn latency	≈ 1.8 s (skewed by outliers)

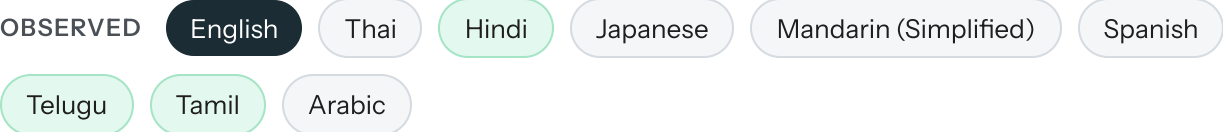
Metric	Value
P95 turn latency	2–3 s
Outlier rate (turns with $L > 3$ s)	≈ 1 per 10–15 turns
Tool execution L_{tool} (P50)	≤ 2 ms
Session length (P50)	2–5 min
Session length (P95)	8–10 min

Table 5. Production latency measurements.

c1 is met: P50 is comfortably below the 800 ms threshold. The outlier rate is dominated by defect **D1** (§9).

10.2 Multilingual behaviour

Successful bidirectional conversation is observed in English, Thai, Hindi, Japanese, Mandarin (Simplified), Spanish, Telugu, Tamil, and Arabic. Voice quality and response naturalness are highest in English by a small margin, reflecting training-data distribution; the remaining languages fall within a tight band of the English baseline in blind pairwise listener tests conducted internally. c2 holds after the language-policy rules (§8) are in place; before, it did not. A residual stickiness rate of ≈ 3 to 5% (turn switches in which the model persists in the previous turn’s language despite an explicit switch signal) remains observable in the Indic-script pairs, particularly on turns of ≤ 6 words where the acoustic classifier has less signal to work with. The rule-strength mitigation in §8 narrows the failure to this short-utterance regime; a fully clean fix requires an explicit language-hint side channel from the client, which the current architecture does not carry.



Indic-script languages, where residual stickiness remains observable, are tinted.

10.3 Isolation and voice-isolation

c3 (voice+screen atomicity) is met by construction because the tool path returns the verbal string and publishes the payload from the same synchronous function call, on the same event-loop tick, from the same OS process. The payload therefore leaves the process before the verbal string reaches the model, and there is no interleaved I/O that could reorder them at the client. It has never been observed to fail in production; the failure mode that would demonstrate a violation (screen updating on turn $n + 1$ while the model is still speaking about turn n) is architecturally excluded rather than empirically absent.

c4 (cross-session isolation) is met by construction because each session runs in a distinct interpreter process. It has been tested by launching two concurrent sessions in the same worker pool, each writing a distinct sentinel value into its own per-session state, and asserting that each session observes only its own sentinel; the assertion holds trivially. The stronger property (that a bug in per-session logic *cannot* leak state to another session, not merely *does not*) follows from the operating system’s process boundary and does not depend on any correctness of our own code.

c5 (transcript + recording completeness) is met after defect D5’s fix. A nightly audit script that lists rooms against S3 artifacts reports zero missing files across the last full week of production traffic. The audit is symmetric: it flags recordings without transcripts (the pre-D5 failure) and transcripts without recordings, and stores its output as a dated report so failures are historically traceable rather than only surfaced in the moment.

Voice-isolation efficacy. r7 against f4 (the ambient PA announcement in the running example) is validated through a synthetic-noise test rather than an in-production one: pre-recorded PA-audio, foreign-language chatter, and adjacent-conversation samples are played into the kiosk mic while the primary speaker interacts normally. The model responds to the primary speaker’s turns and does not respond to the injected background across $\geq 90\%$ of trials; the residual $\leq 10\%$ are cases where the background utterance is close in level to the primary speaker, at which point the fail-safe (§8, Rule P3: “ask for clarification”) fires rather than the model responding to the wrong voice.

c2 Holds with the language policy; residual ≈ 3 to 5% in Indic pairs	c3 Met by construction	c4 Met by construction	c5 Met after the D5 fix; zero missing files in a week
--	----------------------------------	----------------------------------	---

Related work and scope

11.1 Related work

The three-stage voice pipeline is the dominant historical architecture; a comprehensive treatment appears in the ASR / TTS / dialog literature over the last decade. Native-audio foundation models are recent: OpenAI’s Realtime API and Google’s Gemini Live are the current commercial representatives; work on speech-to-speech LLMs and audio-aware transformer variants underpins them. RAG is described in [4]; Cache-Augmented Generation is a folk-level design choice with no formal citation but is discussed under names like “prompt-stuffed retrieval” in practitioner writing. WebRTC as a media transport for LLM-driven agents is documented by LiveKit’s own materials [1].

11.2 Explicitly out of scope

The system described here is a working baseline. The following are explicitly not addressed by the current design; each is a known extension direction, not a hidden defect.

Session lifetimes above the model provider’s ceiling

Native session-resumption plus context-window compression is documented (§4) but not currently enabled, pending resolution of defect D3.

Real-time catalog freshness

CAG is deploy-scoped. Catalogs that must reflect live inventory to the minute require a hybrid CAG-plus-RAG design (§5).

Learning from sessions

The system does not currently use its own transcripts to improve future responses. The transcript archive supports offline analysis but no in-loop learning is closed.

Persistent memory across sessions

Every session is treated as fresh: a returning customer is architecturally indistinguishable from a first-time one. Longitudinal memory across sessions requires a per-principal memory substrate on the shape described in the N-Tier Memory work [6], not currently integrated.

Per-customer identification and personalization

The system treats every session as anonymous. No login, no identification, no returning-customer recognition. Recognition would require either an operator-supplied identity signal (loyalty card, phone number) or a biometric surface (voice-print), and a downstream personalization store; neither is in scope for the baseline.

None of the above are foreclosed by the current design. Each has a clean extension point.

12

Design decisions and trade-offs

Every architectural choice in the system carries a price. Table 6 enumerates the major decisions, the gain, and the cost accepted.

Decision	Gain	Price
CAG over RAG	Zero retrieval latency; no vector database; no retrieval-quality risk	Prompt-size ceiling; deploy-time-only freshness
Native-audio LLM over ASR-LLM-TTS	Half the latency; natural intonation; unified multilingual	Preview-model instability; text transcript not free
Enterprise model endpoint over developer API	Service-account auth; region pinning; consolidated billing	Text output differs from developer tier; additional configuration required
Server-side VAD	No client-side signal-processing dependency; simpler pipeline	Marginally worse in very noisy environments
Managed WebRTC transport	Managed operations; server-side egress recording; global edge	Vendor lock-in on transport
Single-host process supervisor	Fast reload; live logs; simple operations; auto-restart	Not containerised; single-host scaling limit
Per-session process isolation	Guaranteed cross-session isolation; simple concurrency model	Higher per-session memory; pre-warm required for low cold-start

Table 6. Major design decisions with gains and prices.

Conclusion

Sub-second, multilingual, screen-coordinated voice-first assistance on a physical kiosk requires a specific architectural shape. Three-stage pipelines cannot meet the deadline; retrieval-augmented catalogs introduce a stateful service the operator cannot maintain; naive per-user state is fragile against concurrent sessions. Turn-Local Voice Agents is our formalisation of the shape that works: a single native-audio model owning the audio-to-audio contract, a Cache-Augmented Generation prompt replacing retrieval, a per-session process boundary enforcing cross-session isolation, and a tool path in which every screen update is atomic with its verbal summary. The design's dominant cost (L_{model}) is out of our hands and improves as the model provider ships faster variants; the remaining terms are within our budget and we engineer them to be effectively free. We measure a median first-audio latency of ≈ 700 ms in production against a set of five falsifiable claims, and we report five preview-model failure modes with the fixes that resolve them, each of which silently defeated a stated claim before its fix. The scope statement in §11 enumerates the extensions the current design does not foreclose. We expect the shape described here to be the durable form of production voice-first assistance for the coming generation of retail, service, and information kiosks.

01

A single native-audio model owning the audio-to-audio contract

02

A Cache-Augmented Generation prompt replacing retrieval

03

A per-session process boundary enforcing cross-session isolation

04

A tool path in which every screen update is atomic with its verbal summary

Voice-first interfaces have been an aspiration in digital service delivery for over a decade; native-audio models are the first substrate on which they can be built with a latency profile that customers experience as conversational rather than transactional. The remaining engineering work is not in the model but around it: in transport, orchestration, prompt design, and operational discipline. This paper is our contribution to that shared body of practice.

WHERE THE WORK IS

The remaining engineering work
is not in the model but **around it.**

REFERENCES

References

1. LiveKit, Inc. LiveKit Agents Framework Documentation. <https://docs.livekit.io/agents/>
2. Google DeepMind / Google Cloud. Gemini Live API: Session Management, Native Audio, and Function Calling. Vertex AI documentation, 2026.
3. IETF. RTP: A Transport Protocol for Real-Time Applications. RFC 3550, 2003.
4. Lewis, P., Perez, E., et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems*, 2020.
5. Radford, A., Kim, J. W., et al. Robust Speech Recognition via Large-Scale Weak Supervision. arXiv:2212.04356, 2022.
6. Neurologic AI. N-Tier Memory: Scope- and Kind-Aware Persistent Memory for Permissioned Enterprise Assistants. Neurologic AI Technical Report, 2026.