

Building a Production Telephony Voice Agent on a Cascaded Pipeline: An Experience Report

Vivek Sinha
Shielva

Abstract

We report on building and operating a multi-tenant telephony voice agent that answers phone and WhatsApp calls with a cascaded speech-to-text, dialog-flow and retrieval, and text-to-speech pipeline. The system serves three ingress paths (a carrier WebSocket media stream, a SIP trunk bridged into a WebRTC media server, and WhatsApp Business Calling) through one shared agent. We describe the architecture and then focus on what went wrong in production and how each problem was root-caused. A baseline turn took over 10 s; the dominant cost was a retrieval service that streamed its answer as a single event, not the dialog-flow hop it had been blamed on (measured at 134 to 291 ms). We describe an acoustic endpointer with a rule-based semantic continuation hint, a block-LMS echo canceller that removes 28.7 dB of synthetic echo at 0.074 ms per 20 ms frame but remains disabled in production pending real-call validation, defects that made calls impossible to end, template variables read aloud to callers, and a class of silent failures in which the call connects but the agent is deaf or mute. We separate measured results from projections throughout, and we list what remains unvalidated.

Keywords: voice agents, telephony, turn-taking, barge-in, acoustic echo cancellation, retrieval-augmented generation, SIP, WebRTC, latency

1. Introduction

Voice agents on the public telephone network have constraints that browser voice demos do not. The carrier leg is narrowband (G.711 at 8 kHz [1]), callers interrupt, echo returns from the far end, and a call that cannot be hung up keeps billing. The agent must also answer from a business's own data, which in our platform means running an author-designed dialog flow and a retrieval-augmented generation (RAG) [2] step on every turn.

This paper is an experience report, not a benchmark study. Our contributions are: (i) an architecture that lets one agent serve three telephony ingress paths; (ii) a latency investigation that overturned an assumption embedded in our own code comments; (iii) practical designs for endpointing, barge-in and echo cancellation on 8 kHz audio; and (iv) a catalogue of failure modes, many of them silent, with their root causes. Where we have only synthetic or offline measurements we say so.

2. System Overview

A voice agent is an ordinary chat bot from our agent platform plus a phone binding that maps a normalised number to the bot, voice, greeting, language and transport. A telephony service owns call handling; a separate core API owns dialog-flow state, guardrails and the allowlist of external actions the flow may call. The telephony service never interprets flow nodes itself. This costs one network hop per turn, which we measured (Section 3).

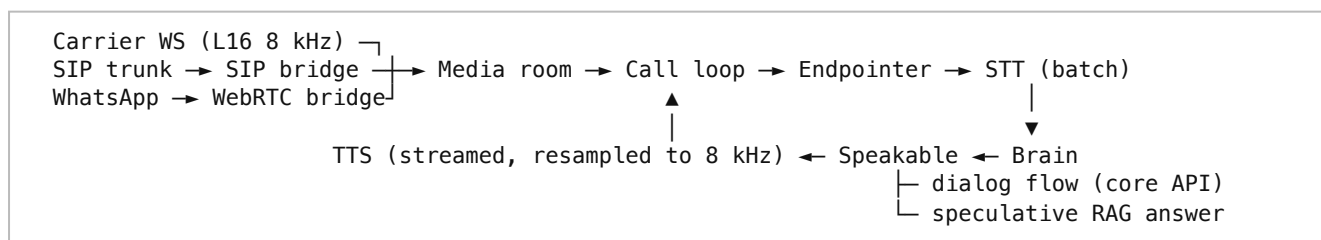


Figure 1. Three ingress paths converge on one call loop. The brain races the dialog flow against a speculative retrieval answer and cancels the loser.

2.1 Ingress paths

Carrier WebSocket. The carrier calls a webhook, receives a stream URL, and sends 16-bit linear PCM at 8 kHz over a WebSocket. Because our API gateway strips WebSocket upgrade headers by design, the media stream needed its own hostname routed directly to the telephony service.

SIP trunk. Carrier SIP [3] terminates on an open-source SIP-to-WebRTC bridge that places each call in a media-server room. A watcher discovers new call rooms by polling every 500 ms rather than by webhook, so no public callback endpoint is needed and calls arriving during a restart are not lost. All carriers are configured as generic SIP; there are no per-carrier provider classes.

WhatsApp Business Calling. Inbound calls first ran over SIP with TLS and SDES-keyed SRTP [4,5]. Outbound calls over the same bridge failed twice: the bridge's TLS ALPN value was rejected, and the provider then returned an SDP validation error because the Opus codec line lacked its channel count. We concluded the SIP route was not viable for outbound and moved to the provider's calling API with a WebRTC [6] peer that joins the same media room, so the same agent answers. That peer holds the connection in process memory and therefore runs as a single replica. Caller audio is discarded until 1.5 s after answer, to drop pre-answer media.

2.2 Media plane

The production pipeline is self-built (about 2,300 lines): an acoustic endpointer, a rule-based semantic turn hint, speculative recognition, barge-in, an echo canceller, and a voice shaper (high-pass filter at 300 Hz, presence EQ, loudness target of -16 dBFS). An alternative built on the Pipecat framework [7] exists behind a configuration switch. On an offline replay of the same recordings it produced 0 suspected cut-offs against 4 for the native stack, and boot-time warm-up cut its cold start from 8.6 s to 0.2 s. It has not yet carried a live call, so production still runs the native stack.

2.3 Recognition, reasoning and synthesis

Speech recognition uses Whisper-family models [8] through batch HTTP APIs; there is no streaming recognition on any path. To recover some of the cost, recognition starts speculatively when the endpointer enters its possible-end state, so the round trip overlaps the hold window. A per-call prompt places known facts, such as a candidate's name, at the front of the recognition prompt.

Each call has one brain object. On each turn it calls the dialog-flow endpoint, which streams typed frames (metadata, scripted speech, reply, done), and in parallel starts a speculative retrieval answer; whichever is not needed is cancelled. Bots without a flow are identified in the first frame, before any model work. Replies pass through a "speakable" filter that strips markup and caps length at 420 characters at a sentence boundary. We added the cap after a reply produced 35 s of audio and the caller hung up at 19 s. We chose this over a second summarising model call, which would add latency on every turn.

Synthesis streams 24 kHz audio from a commercial TTS service, resampled to 8 kHz. Greetings and scripted flow lines are cached in object storage under a content address of text, voice, engine, rate and language, and can be pre-rendered before a campaign.

3. Latency: Diagnosing a 10-Second Turn

The first production turn we instrumented took 10,173 ms: 217 ms of recognition and 9,956 ms attributed to the language model. A code comment claimed the dialog-flow hop cost 1.78 s, and it was the first suspect. Table 1 shows the per-stage measurements.

Table 1. Measured per-turn costs before the fix.

Stage	Measured
Dialog-flow hop (core API)	134 to 291 ms
Retrieval answer, batched	1,562 to 2,965 ms
Retrieval "stream": first token / done	1,610 ms / 1,645 ms, one event
Gap between first and last sentence delivered	0 ms

STT / TTS per turn	218 to 378 ms / 181 to 412 ms
Two model calls per turn vs one	13,924 ms vs about 7 s

The flow hop was cheap; the comment was stale. The real problems were two. First, the retrieval service's streaming endpoint waited for the complete answer and emitted it as a single event, so "streaming" gave no head start (the 0 ms first-to-last gap is the signature). Second, a text-only stream flag silently skipped retrieval, so a question about clinic hours received a generic greeting, and to compensate the pipeline was issuing a second, batched call. We removed the duplicate call, made one grounded batched request, and split its answer into sentences locally so synthesis starts on the first sentence. We kept answer streaming off in production, choosing grounding over speed, until token streaming that keeps retrieval enabled is verified in production.

A second investigation, on a structured screening call, found first turns of 11 to 24 s. We tested prompt size as a cause and ruled it out (98-character prompt: p50 626 ms; 663-character prompt: p50 618 ms). The cause was an opening node that ran a wasted classification on every call. After the fix, first turns took about 750 to 950 ms and later turns 1.6 to 2.9 s. Model-arbitrated classification ran at p50 648 ms; a 3 s timeout clipped roughly 1 run in 9, so we raised it to 5 s.

4. Turn-Taking

The endpointer is an acoustic state machine (listening, speaking, possible end, waiting). With a 620 ms silence budget it committed turns while callers paused mid-thought, yielding transcripts of 4 to 15 characters. Raising the budget to 1,500 ms fixed the fragments but added latency to every turn. The root cause was that the semantic continuation hint, a rule-based check for unfinished utterances, was wired only on the WebSocket path, not on SIP. With the hint wired on both, the base budget returned to 1,050 ms, and utterances that still read as unfinished may hold up to a further 3.5 s. The hint may delay a commit but never veto one, so it cannot strand a caller. We chose rules over a learned turn model [9] to avoid an extra round trip and a new failure mode.

5. Echo and Barge-In

Barge-in lets a caller interrupt the agent. On the WebSocket path it was initially dead code: turn finalisation was awaited inline in the socket read loop, so no audio was read while the agent was speaking. Running each turn as a separate task fixed it. The SIP path initially had no barge-in at all; when we added it, an 800 ms pre-roll buffer preserved the caller's first words. Long scripted lines over 170 characters are split into sentences with 260 ms gaps so callers have room to cut in.

On a real line, barge-in fired zero times during a 12 s agent sentence. Echo returned about 15 dB below playout, and our margin had been set from a meter that read about 2x high. Because echo returns at a ratio of playout level, no fixed threshold separates caller speech from echo. Table 2 shows the thresholds we settled on.

Table 2. Barge-in thresholds.

Setting	Canceller not converged	Canceller converged
Margin below playout	13 dB	3 dB
Sustain	320 ms	160 ms
Lockout after playout start	550 ms	200 ms
Absolute floor	-44 dBFS	-44 dBFS

To remove echo by subtraction instead of thresholding, we built a block-LMS adaptive filter [10]. Bulk delay is estimated by FFT cross-correlation over an 800 ms window, double-talk is detected against the reference signal rather than the residual, and a divergence guard discards the filter if it adds energy. On synthetic 8 kHz audio with a 120 ms echo path it removed 28.7 dB at 0.074 ms per 20 ms frame (0.204 ms per frame in the production container). Step size mattered: $\mu = 0.02$ gave 25 dB, 0.05 gave 28 dB, and 0.2 diverged.

The canceller is disabled in production. It was tuned on 160-sample frames, but one carrier delivers audio in batches of about 800 ms, and the SIP call loop does not yet use it. SIP barge-in therefore runs on the conservative thresholds.

Real-call echo return-loss measurement is open work.

6. Ending Calls

Callers could not end calls. The flow's retry budget is per question, so "I want to stop" was treated as an invalid answer and re-asked. We added a stop verdict to the answer checker, guarded so that a negative answer ("no") is not read as a stop. Two runtime defects then had to be fixed before any ending worked: a completion flag was reassigned, and a retry tail ran unconditionally. Completion and hand-off flags had always been present on the reply frame, but the telephony side never read them. Hang-up mechanics differ by transport. On SIP we delete the media room; disconnecting the agent alone leaves the carrier leg up and billing. On the WebSocket carrier, which has no hang-up API, we close the receive loop. A hand-off to a human is not a hang-up; only flow completion ends the call.

7. Template Variables Read Aloud

Flow authors write lines such as "Hello {{name}}". The substitution function leaves an unknown variable verbatim, and on one call the synthesiser read the braces aloud. The root cause was not substitution. Three automation products each kept their own list of declared call variables, and the lists had drifted from what was actually sent: one offered authors 9 variables while sending 20, including 8 of its 10 spoken lines, and another declared variables nothing sent. We replaced the three lists with one shared declaration, a registry, and a parity test that fails the build when declarations and payloads diverge. We deliberately did not add a runtime sanitiser that would hide the mismatch.

8. Multilingual Calls

The language code "en" was hard-coded in four places. We replaced it with a per-call language object whose mode is unset, a fixed ISO 639-1 code [11], or auto-detect (greeting in English). The first release of auto-detect never switched language: the recogniser returns language names ("chinese"), not codes, so every comparison failed silently. We added name-to-code normalisation. Engine coverage differs: the TTS service supported 41 languages when probed live, while the recogniser detects about 99, so auto mode filters against what the synthesiser can speak. Several languages, including Urdu, Odia and Assamese, were not supported by the TTS service, which matters for Indian deployments. Pinning the wrong language produces fluent nonsense rather than an error.

9. Silent Failures

The most expensive defects produced no error: the call connected and the agent was deaf or mute. Examples: an unset retrieval endpoint produced a canned reply; a blank voice identifier caused a TTS 400 that was swallowed; a 24 kHz versus 16 kHz resampling mismatch would have played audio 1.5x fast; a default value in an attribute lookup hid a wrong field name, so nothing was ever transcribed; an RMS scale mismatch held the speech gate open on every frame; event handlers registered after connecting left the call deaf; and a missing WebSocket accept left a call answered and silent. A related measurement lesson: one carrier's stored call recordings are normalised, so they are not evidence of what we sent. We now measure our own output. We also fixed a 25% recognition billing over-count caused by rounding up a fixed 800 ms buffer.

The common thread is defaults that hide absence. We now treat a missing configuration value, an empty identifier or an unknown field as a failure at start-up, not a fallback at call time.

10. Limitations and Open Work

- The echo canceller is validated only on synthetic echo and is disabled in production.
- The Pipecat-based pipeline passed an offline replay gate but has not run a live call.
- Recognition is batch only; streaming recognition is not implemented.
- Token streaming from the retrieval service with retrieval enabled is not verified in production.
- Outbound WhatsApp calling is blocked on provider eligibility; inbound via the WebRTC bridge is untested.
- Latency figures come from instrumented individual calls and replays, not a controlled study with confidence intervals.

11. Conclusion

A cascaded voice agent on the telephone network is mostly an exercise in measuring the right thing. Our largest latency gain came from discovering that a "streaming" dependency was not streaming and that a stage blamed in our own comments was cheap. Turn-taking and barge-in needed thresholds derived from real lines, not meters or recordings. Many of our defects were silent, and the fix was to make absence fail loudly. We hope this account is useful to teams building similar systems.

References

1. ITU-T Recommendation G.711. Pulse code modulation (PCM) of voice frequencies. International Telecommunication Union, 1988.
2. P. Lewis et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems* 33, 2020.
3. J. Rosenberg et al. SIP: Session Initiation Protocol. RFC 3261, IETF, 2002.
4. M. Baugher et al. The Secure Real-time Transport Protocol (SRTP). RFC 3711, IETF, 2004.
5. F. Andreassen, M. Baugher, D. Wing. Session Description Protocol (SDP) Security Descriptions for Media Streams. RFC 4568, IETF, 2006.
6. H. Alvestrand. Overview: Real-Time Protocols for Browser-Based Applications. RFC 8825, IETF, 2021.
7. Pipecat: open-source framework for voice and multimodal conversational AI. Daily, <https://github.com/pipecat-ai/pipecat>.
8. A. Radford et al. Robust speech recognition via large-scale weak supervision. arXiv:2212.04356, 2022.
9. G. Skantze. Turn-taking in conversational systems and human-robot interaction: A review. *Computer Speech & Language* 67, 2021.
10. S. Haykin. *Adaptive Filter Theory*, 5th ed. Pearson, 2014.
11. ISO 639-1:2002. Codes for the representation of names of languages, Part 1: Alpha-2 code. International Organization for Standardization.