Facsímil 11

IA multimodal

Capítulo 07

Audio, voz y conversación en tiempo real

686f6c61 · 2026

Capítulo 07: Audio, voz y conversación en tiempo real

Qué deberías poder hacer al terminar

La voz parece mágica porque se parece a hablar con otra persona. Pero por dentro no es magia: es señal, red, transcripción, detección de turno, razonamiento, herramientas, síntesis, reproducción, permisos, logs y evaluación. Si una de esas piezas falla, el agente no “habla raro”; interrumpe mal, ejecuta una acción que no debía, guarda datos personales en claro o deja al usuario esperando en silencio.

En las slides del workshop aparece la idea general: pipeline clásico `STT + LLM + TTS` frente a sistemas speech-to-speech o APIs realtime. Aquí vamos más despacio. El objetivo no es que sepas nombrar una API. El objetivo es que puedas mirar una arquitectura de voz y hacer preguntas de ingeniería.

Al terminar este capítulo deberías poder hacer esto:

RESULTADO DE APRENDIZAJE	EVIDENCIA DE QUE LO SABES HACER
Explicar qué es audio digital.	Distingues muestra, frecuencia, frame, códec y latencia.
Separar ASR, LLM, tools y TTS.	No confundes transcripción con intención ni intención con permiso.
Diseñar un contrato de turno.	Guardas transcripción, timestamps, decisión, evidencias, límites y métricas.
Medir un agente de voz.	Calculas WER, latencia de primera voz, endpointing y barge-in.
Elegir transporte y arquitectura.	Sabes cuándo usar WebRTC, WebSocket, proveedor realtime o pipeline propio.
Proteger acciones y datos.	No ejecutas tools peligrosas ni guardas PII solo porque venía en audio.
Practicar en el cuaderno del facsímil.	Lo abres en Google Colab, ejecutas las celdas, miras las tarjetas de turno y cambias una política para ver cómo cambia la decisión.

La frase central del capítulo:

> Un agente de voz no es “un chatbot con micrófono”. Es un sistema de tiempo real donde la incertidumbre de audio afecta permisos, latencia, privacidad y experiencia.

La escena: una llamada de soporte que parece sencilla

Imagina una oficina universitaria. Una persona llama y dice:

> “Quiero saber si puedo enviar la beca aunque el justificante siga pendiente.”

Eso parece una pregunta normal. Ahora añade realidad:

PROBLEMA	QUÉ OCURRE	QUÉ NO DEBE HACER EL SISTEMA
Ruido	El ASR oye “color” donde el usuario dijo “correo”.	Cambiar datos personales por una transcripción dudosa.
Pausa	El usuario respira y el sistema cree que terminó.	Cortar el turno demasiado pronto.
Interrupción	El agente empieza a hablar y el usuario dice “espera”.	Seguir soltando audio como un IVR antiguo.
Tool peligrosa	El usuario dice “cancela mi matrícula si no está pagada”.	Ejecutar una acción irreversible sin confirmación.
Datos sensibles	El usuario dicta DNI, teléfono o número de tarjeta.	Guardar trazas con datos en claro.

La voz amplifica los fallos porque el usuario no está mirando un JSON. Si tarda, se nota. Si interrumpe y no paras, se enfada. Si entiende mal una palabra, puede cambiar una decisión. Si una tool actúa mal, el problema ya no es “la respuesta fue mala”; el sistema hizo algo.

Lectura de ingeniería: una conversación es un sistema distribuido en miniatura

Un agente de voz parece una única experiencia, pero por debajo combina componentes con ritmos distintos. El micrófono captura frames, el VAD decide si alguien habla, el ASR produce hipótesis, el LLM decide intención, las tools consultan sistemas, el TTS sintetiza audio y el reproductor lo emite. Cada etapa añade latencia y cada etapa puede fallar. Por eso la voz exige pensar como en sistemas distribuidos: eventos, estados, timeouts, reintentos y degradación.

La transcripción tampoco es una verdad. Es una observación probabilística. Si el usuario dicta un número de expediente, una fecha o un apellido, un pequeño error puede cambiar el resultado. En una conversación casual quizá se corrige hablando. En una acción administrativa, financiera o sanitaria, necesitas confirmaciones, lectura de vuelta, slots con confianza y reglas que bloqueen acciones cuando la evidencia oral no es suficiente.

El contrato de turno

En texto, un turno suele ser un mensaje. En voz, un turno es una negociación. El sistema debe decidir cuándo empieza la voz, cuándo termina, si hubo interrupción, si el audio fue suficiente, qué transcripción es estable, qué parte era ruido, qué parte era una corrección y qué estado conversacional queda. Si ese contrato no existe, el agente puede responder demasiado pronto, pisar al usuario, ejecutar una tool incompleta o repetir información ya corregida.

Un contrato de turno útil incluye `utterance_id`, timestamps, transcript parcial y final, confianza por slot, intención candidata, slots críticos, acción propuesta, confirmación requerida y motivo de bloqueo. Para una demo, esto parece excesivo. Para producción, es lo que permite explicar por qué no se ejecutó una transferencia, por qué se pidió repetir un dato o por qué se escaló a humano.

Latencia como requisito, no como molestia

La experiencia de usuario y la seguridad se tocan. Un sistema que tarda demasiado invita a interrumpir. Un sistema que no entiende barge-in genera frustración. Un sistema que ejecuta tools por una transcripción dudosa es peligroso. Por eso medir solo WER es insuficiente. Hay que medir latencia de primer audio, latencia de turno completo, cortes prematuros, interrupciones, acciones canceladas y tasa de revisión humana.

También hay que diseñar degradación. Si el ASR baja de confianza, quizá el sistema debe pedir repetición. Si una tool tarda demasiado, quizá debe responder con estado parcial. Si el TTS falla, quizá debe mostrar texto. Si la red se corta, quizá debe cerrar sin ejecutar acciones. Una conversación operable no es la que nunca falla; es la que falla de manera comprensible y segura.

Voz, privacidad y acciones

La voz trae información que el usuario no siempre sabe que entrega: acento, entorno, ruido de fondo, otras personas, tono emocional y datos biométricos potenciales. Si además la conversación llama tools, el riesgo crece. Un transcript puede ser suficiente para el caso de uso; conservar audio bruto durante meses puede no estar justificado. Un embedding de voz puede ser sensible aunque la respuesta final no lo parezca.

La práctica correcta no pregunta “¿habla natural?”. Pregunta: qué estado conserva cada turno, qué evidencia justifica cada tool, qué se guarda redactado, qué se descarta y qué ocurre cuando el audio no permite decidir. Esa es la diferencia entre una demo simpática y un sistema de voz operable. Un alumno debería poder entregar un log de sesión donde se vean eventos, latencias, slots, confirmaciones y bloqueos. Si solo entrega una transcripción bonita, todavía falta ingeniería.

Audio desde cero: lo mínimo que hay que saber

El sonido es una señal continua. Un ordenador no guarda continuidad infinita; guarda muestras. Si la frecuencia de muestreo es f_s, una señal continua $x(t)$ se convierte en una secuencia discreta:

$$x[n] = x\left(\frac{n}{f_s}\right)$$

SÍMBOLO	SIGNIFICADO
$x(t)$	Señal continua en el tiempo.
$x[n]$	Muestra discreta número n.
f_s	Frecuencia de muestreo, por ejemplo 16 kHz o 48 kHz.
n/f_s	Instante temporal de la muestra.

En voz telefónica y ASR es común trabajar con 16 kHz mono porque la voz humana útil para reconocimiento cabe bien ahí. En WebRTC y audio moderno puedes encontrar 48 kHz, Opus, cancelación de eco, supresión de ruido y control de ganancia. Para ingeniería no basta con decir “audio”: hay que saber qué formato entra.

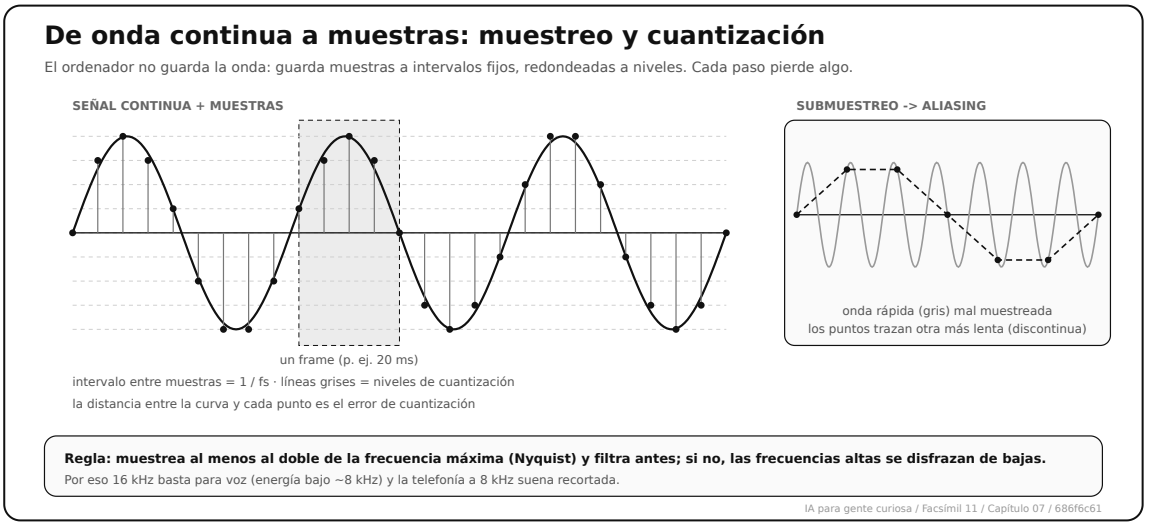
Por qué 16 kHz: el teorema de muestreo

Que se elija 16 kHz no es un capricho. El teorema de muestreo de Nyquist-Shannon dice que, para reconstruir sin pérdida una señal cuyo contenido en frecuencia llega hasta $f_{\max}$, hay que muestrear al menos al doble de esa frecuencia:

$$f_s \geq 2f_{\max}$$

SÍMBOLO	SIGNIFICADO
f_s	Frecuencia de muestreo.
$f_{\max}$	Frecuencia más alta presente en la señal.
$2f_{\max}$	Frecuencia de Nyquist: el mínimo para no perder información.

La energía de la voz humana útil para reconocimiento se concentra por debajo de unos 8 kHz, así que 16 kHz cubre esa banda con margen.[1] Si muestreas por debajo de $2f_{\max}$, las frecuencias altas no desaparecen: se disfrazan de frecuencias bajas que nunca estuvieron. Ese artefacto se llama aliasing, y por eso los sistemas reales aplican un filtro antialiasing antes de muestrear. La telefonía clásica trabaja a 8 kHz, banda estrecha, y suena “metálica” precisamente porque recorta por encima de 4 kHz; los 48 kHz habituales en WebRTC dan margen de sobra para voz y música.



El audio digital nace de dos aproximaciones: muestrear en el tiempo (frecuencia de muestreo) y cuantizar en amplitud (niveles). El teorema de Nyquist fija el mínimo de muestreo; por debajo aparece aliasing, una onda que el sistema "oye" pero que no existía.

CONCEPTO	QUÉ SIGNIFICA	POR QUÉ IMPORTA
Sample rate	Muestras por segundo.	Afecta calidad, coste, compatibilidad y resampling.
Bit depth	Precisión de cada muestra.	PCM 16-bit es habitual en pipelines simples.
Frame	Trozo pequeño de audio, por ejemplo 20 ms.	ASR/VAD/streaming trabajan por frames.
Códec	Forma de comprimir/transmitir audio.	Opus está pensado para voz interactiva y baja latencia.[2]
Jitter	Variación en llegada de paquetes.	En conversación se nota como cortes, esperas o audio irregular.
Resampling	Convertir de una frecuencia a otra.	Puede meter coste y artefactos si se hace mal.

Un error muy común: tratar audio como si fuese un archivo que subes y ya está. En tiempo real, el audio llega por trozos. Cada trozo debe viajar, decodificarse, filtrarse, entrar en VAD, alimentar ASR y mantener una conversación viva.

Pipeline clásico: STT + LLM + TTS

La arquitectura más fácil de entender es esta:

1. Capturas audio del usuario.
2. Transcribes con ASR.
3. Pasas texto al LLM.
4. El LLM decide o llama a herramientas.
5. Generas texto de respuesta.
6. Sintetizas voz con TTS.
7. Reproduces audio.

Funciona. De hecho, en muchos productos sigue siendo la opción más controlable. Pero suma latencias:

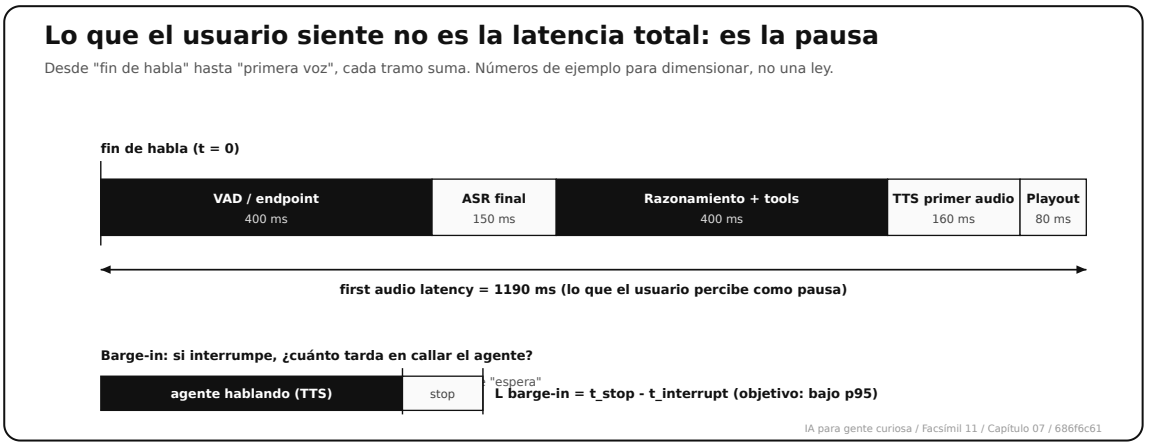
ETAPA	QUÉ AÑADE	QUÉ PUEDE FALLAR
Captura	Permisos, micrófono, navegador, formato.	No hay audio, eco, ganancia mala.
VAD/endpointing	Saber cuándo empieza y termina el turno.	Cortar pronto o esperar demasiado.
ASR	Convertir voz a texto.	Errores por ruido, acento, nombres propios.
LLM	Razonar, recuperar, llamar tools.	Alucinar, llamar herramienta incorrecta.
TTS	Convertir texto a voz.	Latencia, pronunciación, tono, prosodia.
Playout	Reproducir audio al usuario.	Jitter, buffer, interrupciones.

Una estimación operativa, no una ley universal:

$$L_{\text{voz}} = L_{\text{captura}} + L_{\text{vad}} + L_{\text{asr}} + L_{\text{razonamiento}} + L_{\text{tts\_first}} + L_{\text{playout}}$$

TÉRMINO	LECTURA PRÁCTICA
L_{captura}	Tiempo hasta tener audio útil.
L_{vad}	Tiempo que tardas en decidir que el usuario terminó.
L_{asr}	Tiempo hasta transcripción parcial/final.
$L_{\text{razonamiento}}$	LLM, RAG, tools, políticas y validaciones.
$L_{\text{tts\_first}}$	Tiempo hasta el primer trozo de audio de respuesta.
L_{playout}	Buffer y reproducción en cliente.

El número que suele doler al usuario no es solo “latencia total”. Es cuánto tarda en oír algo después de terminar de hablar. Si el sistema responde perfecto pero con una pausa incómoda, la experiencia se rompe.



El presupuesto de latencia descompone la pausa que sufre el usuario. Optimizar "la latencia total" sin mirar el primer audio ni el corte por barge-in mejora números que el usuario no siente y deja intactos los que sí.

Pipeline nativo speech-to-speech

Las APIs realtime modernas permiten enviar audio y recibir audio de forma bidireccional, a veces con transcripción, eventos, tool calls y gestión de turnos en la misma sesión. La documentación de OpenAI describe sesiones Realtime donde el cliente se conecta a `/v1/realtime`, envía audio o texto y escucha respuestas, llamadas a herramientas y eventos de sesión; para agentes de voz en navegador recomienda partir de WebRTC y Agents SDK.[3] Gemini Live se presenta como API en preview para interacciones de baja latencia con voz y visión, procesando streams continuos de audio, imagen y texto.[4]

La tentación es decir: “entonces uso nativo y me olvido”. No. Cambia el sitio donde vives la complejidad.

ARQUITECTURA	VENTAJA	RIESGO
STT + LLM + TTS	Mucho control, piezas sustituibles, eval por etapa.	Más latencia y más integración.
Speech-to-speech nativo	Menos fricción conversacional, interrupciones más naturales.	Menos observabilidad por etapa si no instrumentas bien.
Híbrido	Puedes usar nativo para conversación y tools/contratos propios para decisiones.	Requiere diseñar bien eventos, trazas y límites.

En un producto serio, yo no preguntaría “¿cuál suena mejor en demo?”. Preguntaría:

PREGUNTA	POR QUÉ IMPORTA
¿Tengo transcripción parcial y final?	Para mostrar, auditar y detectar errores.
¿Puedo cancelar generación y TTS al interrumpir?	Para barge-in real.
¿Dónde quedan tool calls y sus argumentos?	Para permisos, revisión y trazabilidad.
¿Puedo redactar PII antes de logs?	Para privacidad y cumplimiento.
¿Qué latencias p50/p95 tengo por región?	Para SLO real, no sensación.
¿Puedo reproducir una conversación?	Para depurar incidentes.

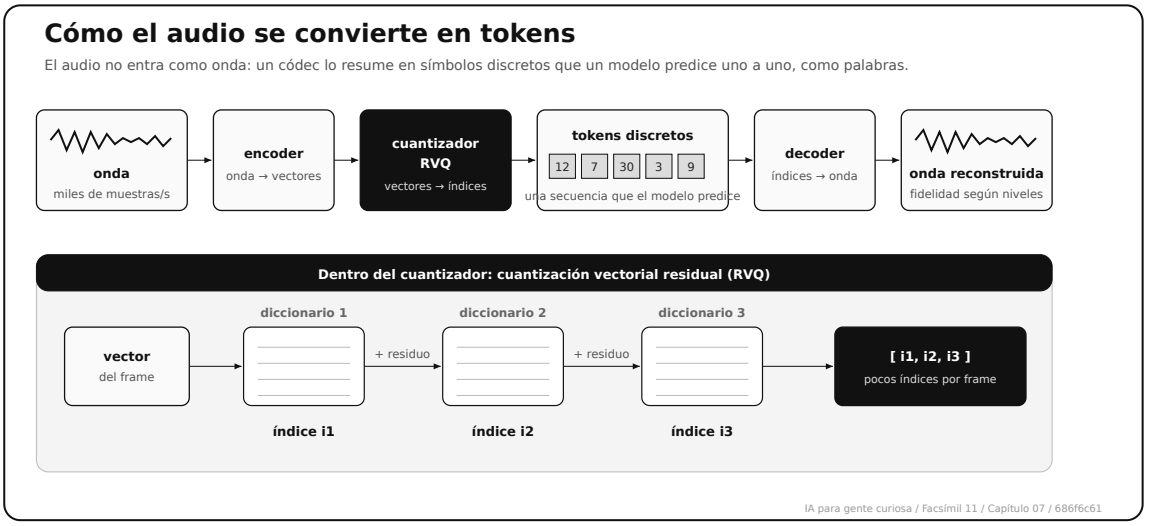
Cómo el audio se convierte en tokens

Hay una pregunta que el pipeline nativo deja en el aire y que conviene no esquivar: si un modelo de lenguaje predice tokens de texto, ¿cómo es que el mismo tipo de modelo puede **generar voz**? La respuesta es la pieza que une este capítulo con la idea central de todo el fascículo. En el capítulo de visión vimos que una imagen no entra entera en el modelo: se trocea en patches y cada patch se convierte en un token. Con el audio ocurre algo equivalente. Una onda de sonido tampoco entra como tal; antes hay que convertirla en una secuencia de **tokens de audio discretos** que el modelo pueda leer y, sobre todo, predecir uno tras otro igual que predice palabras.

La herramienta que hace esa conversión es un **códec neuronal de audio**. A diferencia de un códec clásico como Opus, que está diseñado para comprimir y transmitir, un códec neuronal aprende a representar el audio como una lista corta de símbolos de un vocabulario finito, de forma que esa lista baste para reconstruir la onda con buena fidelidad. SoundStream fue uno de los primeros en mostrarlo de extremo a extremo: un codificador convierte el audio en vectores, un cuantizador los aproxima a entradas de un diccionario aprendido y un decodificador reconstruye el sonido, todo entrenado a la vez.[5] EnCodec extendió la idea con mayor fidelidad y a varios anchos de banda, y se convirtió en una base habitual para sistemas generativos de audio.[6]

El truco que hace viable todo esto se llama **cuantización vectorial residual**: en lugar de un único diccionario gigante, se aplican varios diccionarios en cascada, donde cada uno corrige el error que dejó el anterior. Así, con unos pocos niveles, un segundo de audio se resume en unas decenas o cientos de tokens en vez de las miles de muestras crudas que vimos al principio del capítulo. Y aquí está la consecuencia que importa: una vez que el audio es una secuencia de tokens, un modelo puede **predecir el siguiente token de voz** exactamente con la misma maquinaria con la que predice la siguiente palabra. Eso es lo que hay por debajo de buena parte de los sistemas speech-to-speech: no «entienden el sonido» por magia, lo tratan como un idioma más con su propio alfabeto.

Para ingeniería, saber esto cambia las preguntas que haces. La tokenización de audio tiene un coste, igual que los tokens visuales: un segundo de voz no es gratis, ocupa contexto y compite con el texto, así que el presupuesto de una conversación larga se mide también en tokens de audio, no solo en minutos. La fidelidad depende de cuántos niveles de cuantización uses, lo que vuelve a ser la misma tensión de siempre entre calidad y coste. Y la latencia de generar audio token a token explica por qué el primer fragmento de voz tarda lo que tarda. No necesitas implementar un códec neuronal para construir producto, igual que no implementas el troceado en patches a mano; pero entender que el audio se vuelve tokens evita la sensación de que el speech-to-speech es una caja mágica y te deja razonar sobre su coste, su latencia y sus límites con los mismos conceptos que ya usabas para el texto y la imagen.



Un códec neuronal convierte la onda en una secuencia corta de índices discretos: el encoder produce vectores, la cuantización vectorial residual los aproxima con varios diccionarios en cascada (cada uno corrige el error del anterior) y el decoder reconstruye el sonido. Una vez que el audio es una secuencia de tokens, un modelo puede predecir el siguiente fragmento de voz igual que predice la siguiente palabra.

Eventos de una sesión realtime

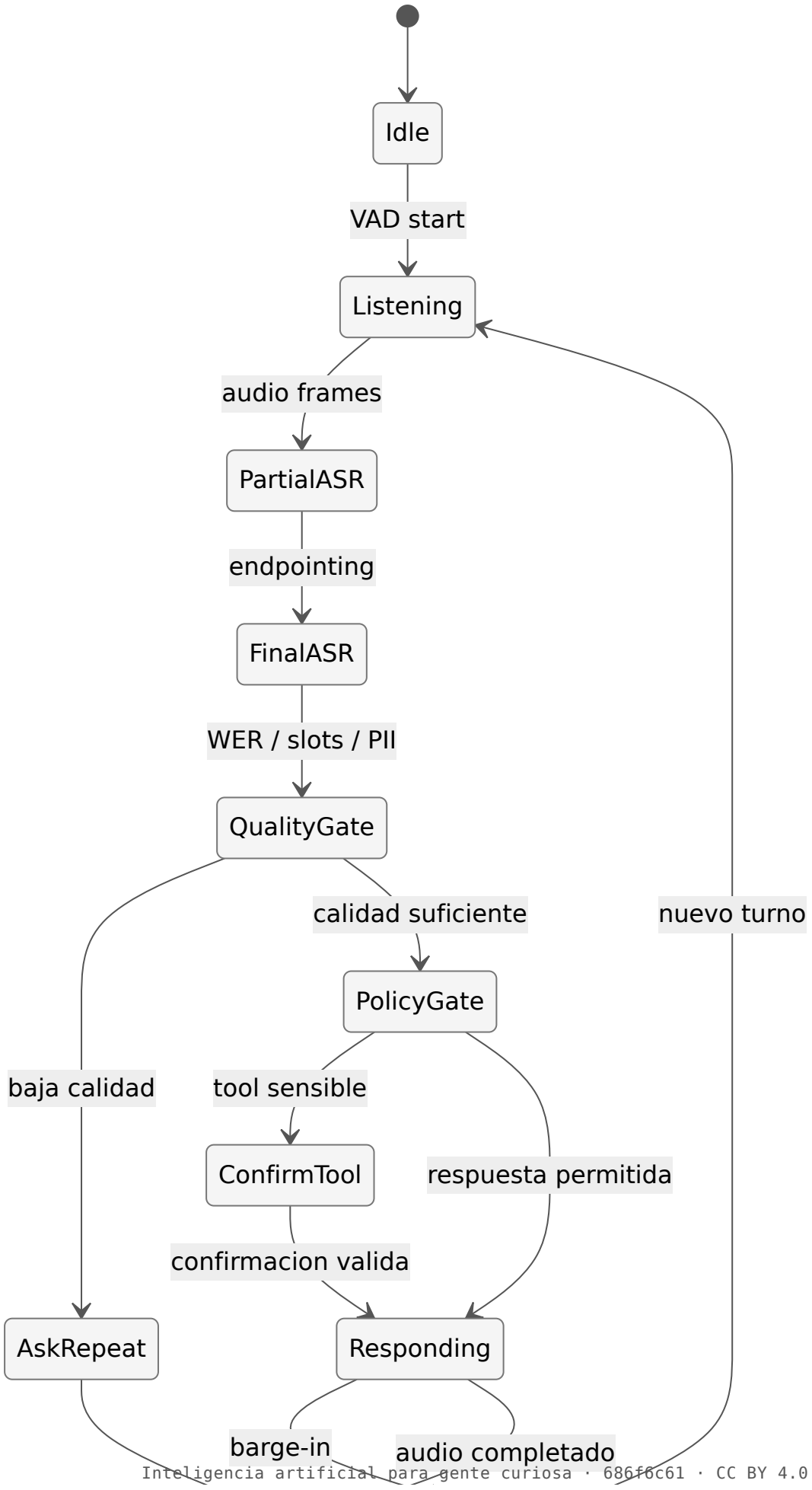
Una sesión de voz no debería modelarse como una petición HTTP gigante. Es una conversación de eventos. OpenAI documenta flujos de conversación Realtime con generación de audio/texto, entrada de imagen, function calling y estado de sesión; también separa el modo conversación del modo transcripción cuando no esperas respuesta del modelo.[7] La referencia de eventos de servidor incluye parámetros de VAD como `interrupt_response`, que permite cancelar una respuesta en curso cuando empieza voz del usuario, e `idle_timeout_ms` para timeouts en modo `server_vad`.[8]

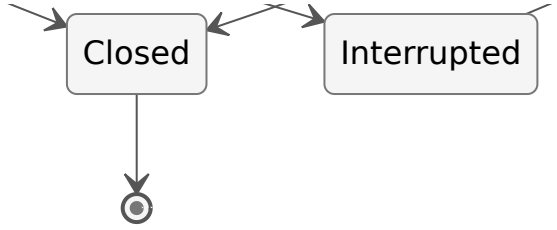
Una forma práctica de pensarlo:

TIPO DE EVENTO	EJEMPLO	QUÉ HARÍA INGENIERÍA
Audio entrante	Frame de PCM/Opus desde cliente.	Validar formato, timestamp y pérdida.
VAD	Empieza habla, termina habla, timeout de silencio.	Abrir/cerrar turno sin ejecutar todavía.
ASR parcial	“quiero cambiar el color...”	Mostrar como provisional, no usar para tool.
ASR final	“quiero cambiar el correo...”	Pasar por gates de calidad y slots críticos.
Respuesta del modelo	Texto/audio delta.	Medir primer delta y primera voz.
Tool call	`cambiar_datos_personales(...)`.	Validar política, permiso y confirmación.
Cancelación	Usuario interrumpe.	Parar TTS y marcar generación anterior como obsoleta.
Trazas	Turno cerrado con métricas.	Guardar lo mínimo necesario y redactado.

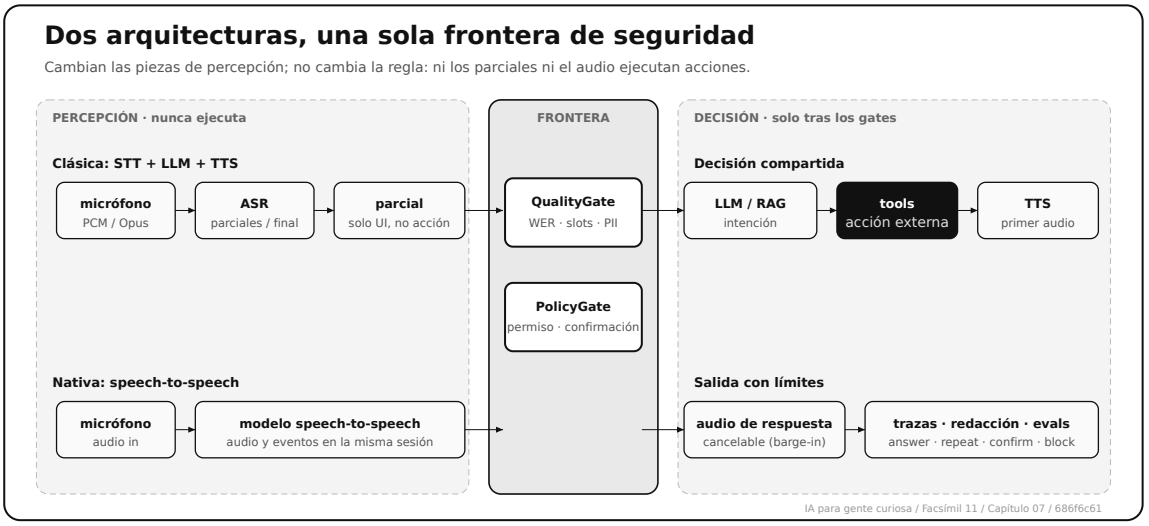
Esto cambia cómo programas. En una app de texto puedes esperar a tener una respuesta completa. En voz, hay decisiones antes, durante y después de la respuesta. La interfaz “natural” solo se sostiene si el backend sabe vivir con estados intermedios.

Ejemplo de máquina de estados para un turno:


Idle
VAD start
Listening
audio frames
PartialASR
endpointing
FinalASR
WER / slots / PII
QualityGate
calidad suficiente
nuevo turno
PolicyGate
baja calidad
tool sensible
ConfirmTool
respuesta permitida
confirmacion valida
AskRepeat
Responding
barge-in
audio completado



El detalle importante: `PartialASR` no debería disparar acciones. Puede servir para UI, captions o prefetch, pero no para cambiar estado. La frontera de seguridad está en `QualityGate` y `PolicyGate`.



> El pipeline clásico y el nativo speech-to-speech reparten la complejidad de forma distinta, pero comparten frontera: los parciales y el audio nunca disparan acciones, y las herramientas solo se ejecutan después del gate de calidad y del de política.

ASR: convertir voz en texto no es entender

ASR significa Automatic Speech Recognition, reconocimiento automático del habla, y su trabajo es más acotado de lo que parece: convierte audio en texto y nada más. No decide la intención del usuario, no concede permisos y no tiene forma de saber si una acción es peligrosa o inocua. Lo único que entrega es una hipótesis textual acompañada de más o menos confianza. Esa distinción, que parece obvia escrita en un papel, es la que más se olvida al programar: en cuanto el texto aparece limpio en la pantalla, la tentación de tratarlo como la verdad es enorme, y ahí empiezan casi todos los problemas de un agente de voz.

No hace falta dominar la historia entera del reconocimiento del habla para construir producto, pero sí ayuda saber de qué familias venimos, porque cada una resolvió una pieza distinta del problema y dejó una idea que sigue viva hoy:

FAMILIA	IDEA TÉCNICA	QUÉ APORTA
CTC	Alinea audio y texto sin alineación fija usando un símbolo en blanco y la pérdida CTC.[9]	Base de muchos ASR: simple y eficaz, aunque asume independencia entre salidas.
RNN-T	Transducción secuencia a secuencia sin alineación previa rígida.[10]	Muy importante para ASR streaming: predice mientras entra audio.
wav2vec 2.0	Preentrena representaciones de audio con aprendizaje autosupervisado y luego ajusta con transcripciones.[11]	Reduce dependencia de grandes corpus etiquetados.
Conformer	Combina convoluciones para patrones locales y Transformer para contexto global.[12]	Arquitectura fuerte en ASR moderno.
Whisper	Entrenado con 680.000 horas de audio multilingüe y multitarea débilmente supervisado.[13]	Robustez y generalización, muy útil como referencia práctica.

De ahí que la salida de un buen ASR no debería ser una cadena de texto pelada, sino una hipótesis rodeada de metadatos que te permitan dudar de ella cuando toca. La confianza, los tiempos y la calidad de audio no son adornos: son lo que distingue "creo que dijo esto" de "esto es lo que dijo":

```
{
  "transcript": "necesito cambiar el correo de contacto",
  "is_final": true,
  "start_ms": 0,
  "end_ms": 2480,
  "confidence": 0.84,
  "language": "es",
  "speaker": "user",
  "audio_quality": "noisy"
}
```

El problema es que muchas APIs y demos te devuelven solo el texto, sin la confianza ni los tiempos, y uno acaba tratándolo como verdad sin darse cuenta. Para preguntas inocuas puede valer: si alguien pregunta el horario y el ASR confunde una palabra, el coste es bajo y la conversación se corrige sola. Para herramientas con efecto, datos personales o decisiones administrativas, la historia cambia: ahí una sola palabra mal entendida puede alterar un correo, un importe o una matrícula, y la confianza del ASR deja de ser un dato curioso para convertirse en parte de la decisión.

Cuando el ASR no se equivoca: se lo inventa

Hasta aquí hemos hablado del error clásico: el sistema oye una palabra y la confunde con otra parecida. Pero hay un fallo más inquietante, y mucho menos conocido, que cambia cómo hay que diseñar la frontera de seguridad. Un ASR moderno no solo sustituye palabras: a veces **transcribe frases enteras que nadie llegó a decir**. No es ruido aleatorio ni un símbolo raro; es texto fluido, gramatical y verosímil, colocado justo donde no había habla. Si un error de palabra es un resbalón, esto es una invención con toda la confianza del mundo.

El sitio donde más aparece es revelador: los segmentos de **silencio, respiración o ruido de fondo**, es decir, justo donde no hay nada que reconocer. Modelos como Whisper se entrenaron con cantidades enormes de audio y subtítulos recogidos de internet, y de ahí arrastran patrones que no tienen que ver con lo que se dijo: rellenan un silencio largo con muletillas típicas de los vídeos que vieron durante el entrenamiento (agradecimientos al espectador, coletillas de despedida, frases de relleno), o repiten en bucle la última frase cuando se quedan sin señal útil. El modelo, ante el vacío, hace lo que mejor sabe hacer un sistema generativo: producir la continuación más probable. Y la continuación más probable de un silencio no es el silencio, es texto.

Un estudio de 2024 sobre transcripciones de Whisper midió este fenómeno de forma sistemática y encontró que una fracción no despreciable de las transcripciones contenía pasajes alucinados que no se correspondían con ningún audio, y que una parte de esos pasajes incluía contenido dañino: autoridad inventada, datos personales fabricados o frases violentas que el hablante nunca pronunció.[14] El trabajo observaba además que las alucinaciones se disparaban con hablantes que hacían pausas más largas, por ejemplo personas con afasia, lo que convierte el problema en algo más que una curiosidad técnica: el sistema falla más precisamente con quien más le cuesta hablar con fluidez.

Para ingeniería, la consecuencia es directa y conecta con todo lo anterior. El VAD no es solo una optimización para ahorrar cómputo: es una defensa contra la alucinación, porque si no mandas al ASR los segmentos sin habla, no le das la ocasión de inventarlos. La confianza por segmento deja de ser un adorno y pasa a ser un filtro: una hipótesis sobre un tramo de baja energía merece más sospecha, no menos. Y la regla de oro del capítulo se vuelve todavía más estricta: si una transcripción parcial sobre un silencio no debería ni mostrarse como provisional, mucho menos disparar una herramienta. Un agente que ejecuta acciones a partir de texto que el ASR fabricó de la nada no tiene un problema de calidad de audio; tiene un problema de seguridad.

WER: medir transcripción sin engañarse

La métrica clásica de ASR es WER, Word Error Rate. NIST SCTK/SCLITE documenta la evaluación comparando una hipótesis de reconocimiento con una referencia mediante alineación y conteo de errores.[15]

$$\mathrm{WER} = \frac{S + D + I}{N}$$

SÍMBOLO	SIGNIFICADO
S	Sustituciones: el ASR cambió una palabra por otra.
D	Deleciones: faltan palabras de la referencia.
I	Inserciones: aparecen palabras de más.
N	Número de palabras de la referencia.

Ejemplo pequeño:

REFERENCIA	HIPÓTESIS	ERROR
“cambiar el **correo**”	“cambiar el **color**”	Sustitución.
“mi expediente **de beca**”	“mi expediente”	Deleción.
“estado de matrícula”	“estado de **mi** matrícula”	Inserción.

WER no lo explica todo. Una WER de 5% puede ser aceptable en una charla, pero fatal si el 5% cae justo en “no”, “cancelar”, “cien” o “mil”. Por eso en sistemas de voz con tools interesa medir por intención, slots críticos y acciones peligrosas, no solo por promedio.

Métricas más allá de WER

Si el capítulo se quedara en WER, estaría incompleto para ingeniería de IA. WER mide palabras. Tu producto falla por decisiones.

Ejemplo de métrica operativa para slots críticos:

$$\mathrm{SlotErrorRate} = \frac{\mathrm{slots\_criticos\_erroneos}}{\mathrm{slots\_criticos\_totales}}$$

SLOT CRÍTICO	POR QUÉ IMPORTA
Negación	“No envíes” frente a “envíes”.
Acción	“Cancelar matrícula” frente a “consultar matrícula”.
Importe	“cien” frente a “mil”.
Identificador	DNI, expediente, pedido, ticket.
Fecha	“hoy” frente a “jueves”.
Entidad	“correo” frente a “color” en el ejemplo del cuaderno.

En un agente de voz con tools yo miraría esta matriz:

MÉTRICA	QUÉ MIDE	CUÁNDO DUELE
WER	Error por palabra respecto a referencia.	Calidad general de transcripción.
CER	Error por carácter.	Nombres, códigos, matrículas, DNI, idiomas con tokenización difícil.
Slot error rate	Campos críticos mal reconocidos.	Tools, formularios, pagos, reservas, acciones.
Intent accuracy	Si la intención se clasificó bien.	Routing y selección de herramienta.
Partial stability	Cuánto cambian los parciales antes del final.	UI en vivo, captions, prefetch y ansiedad del usuario.
Endpoint delay	Tiempo desde fin de habla hasta ASR final o cierre de turno.	Sensación de pausa torpe.
False cut rate	Turnos cerrados antes de que el usuario termine.	Frases largas, dudas, usuarios no expertos.
Barge-in stop latency	Tiempo en detener salida al interrumpir.	Conversación natural.
DER	Error de diarización: quién habló y cuándo.	Reuniones, llamadas con varios hablantes.

La diarización no siempre aparece en un agente de voz sencillo, pero entra rápido cuando hay llamadas, reuniones, operadores o conversaciones con terceros. Pyannote.audio es un toolkit abierto para diarización con bloques entrenables para VAD, cambio de hablante, solapamiento

y embeddings de hablante.[16] La métrica DER suele descomponerse en falsa alarma, habla perdida y confusión de hablante:

$$\mathrm{DER} = \frac{\text{false alarm} + \text{missed detection} + \text{confusion}}{\text{total}}$$

No hace falta meter diarización en todos los productos. Hace falta saber cuándo la necesitas. Si el sistema atribuye “sí, autorizo” a la persona equivocada, tienes un problema de ingeniería y de responsabilidad.

Dataset de evaluación de voz

No evalúes un agente de voz solo con tus tres frases favoritas en la oficina. Los corpus públicos ayudan a comparar ASR y entrenar intuición, pero tu release necesita un set propio de dominio.

DATASET O SET	QUÉ APORTA	QUÉ NO RESUELVE
LibriSpeech	Corpus de unas 1000 horas de lectura en inglés a 16 kHz, derivado de audiolibros, usado durante años en ASR.[17]	No representa llamadas con ruido, acentos locales, nombres de tu dominio ni tools.
Common Voice	Corpus multilingüe de voz recogida y validada por crowdsourcing.[18]	Puede no cubrir tus términos técnicos ni tu canal de audio.
Set interno de dominio	Frases reales o sintéticas revisadas: becas, pagos, incidencias, permisos.	Debe documentarse, versionarse y proteger datos.
Set adverso práctico	Ruido, interrupciones, negaciones, importes, PII, tools peligrosas.	No sustituye tráfico real monitorizado.

Una matriz mínima para tu propio set:

DIMENSIÓN	EJEMPLOS QUE DEBERÍAS INCLUIR
Canal	Micrófono portátil, móvil, auriculares, telefonía.
Ruido	Oficina, calle, eco, teclado, otra voz de fondo.
Idioma/acento	Español de varias regiones, mezcla con inglés técnico si ocurre.
Tarea	Consulta, formulario, tool de lectura, tool de escritura.
Riesgo	PII, negación, importes, cancelación, permisos.
Experiencia	Pausas largas, interrupciones, correcciones del usuario.

La práctica del capítulo incluye `output/voice_eval_matrix.csv` precisamente para esto: comparar WER, slots críticos, estabilidad de parciales y decisión final. Si el sistema oye “color” donde debía oír “correo”, WER ya avisa, pero el slot crítico deja claro por qué no se debe ejecutar nada.

VAD y endpointing: cuándo termina un turno

Dos piezas pequeñas deciden buena parte de la sensación de naturalidad. El VAD, detección de actividad de voz, separa los frames que llevan habla de los que solo llevan silencio o ruido. El endpointing va un paso más allá: a partir de esa señal decide el momento en que el usuario ha terminado su turno y el sistema puede empezar a responder. Suena a fontanería interna, pero es justo la frontera entre una conversación que fluye y un sistema que pisa o que se queda esperando con un silencio incómodo. Una persona sabe cuándo callar por contexto, entonación y respiración; una máquina solo tiene energía, tiempo y umbrales, y con eso tiene que adivinar lo mismo.

La versión más simple del endpointing es una regla de energía y silencio. No es un algoritmo universal ni lo último en investigación, pero sirve para ver de dónde nacen los aciertos y los cortes prematuros:

$$\text{cerrar\_turno} = (\text{energia} < \theta) \land (\text{silencio\_continuo} > 500\,ms)$$

PIEZA	QUÉ CONTROLA
energia	Si el frame parece habla o silencio.
θ	Umbral de energía. Si es alto, pierdes habla baja. Si es bajo, metes ruido.
Silencio continuo	Cuánto esperas antes de cerrar.

Si cierras con 200 ms de silencio, cortas frases normales. Si esperas 1500 ms, el sistema parece lento. El valor correcto depende de idioma, canal, ruido, usuario y tipo de tarea. Un agente de soporte puede tolerar más espera; un asistente de cocina manos libres quizá necesita responder antes.

Turn manager: la pieza que muchas demos esconden

Si el VAD y el endpointing deciden cuándo, el gestor de turno decide qué hacer con cada cosa que llega. Es la pieza que casi ninguna demo enseña, porque en una demo todo sale bien: el usuario habla claro, no interrumpe y no pide nada peligroso. En producción, en cambio, el gestor de turno es quien resuelve, evento a evento, si actualizar la hipótesis, esperar un poco más, pedir que repitan, cortar el audio o frenar una herramienta. Sin esa pieza, la conversación se sostiene en la suerte de que nada raro ocurra. Esta es su tabla de decisiones sanas:

EVENTO	DECISIÓN SANA
Llega audio parcial.	Actualizar hipótesis, no ejecutar.
VAD detecta silencio breve.	Esperar si la frase parece incompleta.
ASR final llega con baja confianza.	Pedir repetición o confirmación.
El usuario interrumpe al TTS.	Cortar audio anterior y cancelar generación si procede.
El LLM propone tool destructiva.	Exigir confirmación y permiso.
Aparece PII.	Redactar antes de logs y minimizar contexto.

Un contrato de turno útil puede tener esta forma:

```
{
  "turn_id": "call-2026-06-14-0007:t12",
  "audio": {
    "sample_rate_hz": 16000,
    "codec": "opus",
    "speech_start_ms": 0,
    "speech_end_ms": 2460,
    "mean_energy": 0.075
  },
  "asr": {
    "transcript": "quiero saber si puedo enviar la beca aunque el justificante siga
pendiente",
    "wer_estimate": 0.08,
    "is_final": true
  },
  "decision": "answer",
  "tool_policy": {
    "tool_name": null,
    "requires_confirmation": false
  },
  "metrics": {
    "endpoint_delay_ms": 400,
    "first_audio_latency_ms": 1190
  },
  "evidence": ["policy_submission_rule", "status_pending_receipt"],
  "limits": ["no decide elegibilidad final"]
}
```


Esto no es burocracia. Es el objeto que te permite depurar, evaluar y explicar por qué el sistema respondió, pidió repetición o bloqueó.

Barge-in: interrumpir no es un detalle de UX

Barge-in es la capacidad de que el usuario interrumpa mientras el agente todavía está hablando. En una conversación humana ocurre continuamente: nos solapamos, nos cortamos, nos corregimos, y nadie lo vive como un fallo. En un producto de voz, sin embargo, si el agente sigue soltando audio cuando la persona ya ha vuelto a hablar, la experiencia se desploma: deja de parecer un interlocutor y pasa a parecer una máquina recitando un guion que no se puede parar. Y lo difícil no es darse cuenta de que el usuario habla, sino reaccionar a tiempo en todas las capas a la vez, porque en ese instante hay audio reproduciéndose, una generación en marcha y un estado conversacional que actualizar.

Un barge-in correcto tiene varias capas:

1. El cliente detecta voz del usuario mientras el TTS está reproduciendo.
2. Se detiene el audio de salida.

3. Se cancela o marca como obsoleta la generación anterior.
4. Se abre un nuevo turno.
5. La traza conserva que hubo interrupción.

Ejemplo realista:

MOMENTO	EVENTO
600 ms	El agente empieza a hablar.
1160 ms	El usuario dice “espera”.
1290 ms	El cliente corta TTS.
2260 ms	ASR finaliza la nueva intención.
2970 ms	El usuario oye la nueva respuesta.

La métrica que miraría:

$$L_{\text{barge-in}} = t_{\text{tts\_stop}} - t_{\text{user\_interrupt}}$$

Esa resta es engañosamente sencilla. Si la latencia de barge-in es alta, la interrupción existe en el diagrama pero no en la experiencia: el usuario dice "espera", el agente sigue hablando medio segundo más y, para cuando calla, ya ha pisado justo lo que la persona intentaba decir. Por eso conviene mirarla en percentiles y no de media, y vigilar el p95: el caso peor es el que el usuario recuerda al colgar.

TTS: hablar no es leer texto

El TTS, síntesis de voz, recorre el camino inverso al ASR: convierte texto en audio. Y aquí también hay más capas de las que se ven desde fuera, porque "leer en voz alta" esconde una cadena de decisiones que el oído da por supuestas. La familia moderna suele separar el problema en etapas, cada una con su propio margen de error:

CAPA	QUÉ HACE
Normalización de texto	Convierte fechas, números, siglas y símbolos en una forma pronunciable.
Representación lingüística	Caracteres, subpalabras, fonemas, acentos, idioma.
Modelo acústico	Produce espectrogramas o representaciones acústicas.
Vocoder	Convierte esa representación en onda sonora.
Streaming	Entrega audio por fragmentos antes de tener toda la frase.

WaveNet mostró generación autoregresiva de audio crudo con gran calidad perceptual para TTS.[19] Tacotron 2 combinó una red secuencia-a-secuencia que predice espectrogramas mel con un vocoder WaveNet.[20] FastSpeech atacó el problema de velocidad y control con generación paralela de espectrogramas.[21] HiFi-GAN mostró vocoders eficientes y de alta fidelidad basados en GANs.[22]

La ingeniería diaria no suele consistir en implementar Tacotron o HiFi-GAN desde cero. Consiste en saber qué duele:

DOLOR	EJEMPLO	CONTROL
Números mal leídos	“1.250” como uno punto dos cinco cero.	Normalización por idioma y dominio.
Nombres propios	Apellidos, campus, productos.	Diccionario/pronunciación custom si el proveedor lo permite.
Latencia	Voz empieza tarde.	TTS streaming, frases cortas, prefetch.
Tono inadecuado	Suena alegre en una incidencia grave.	Estilo, SSML o selección de voz.
Barge-in	El audio no se puede cancelar.	Playout controlado y eventos de cancelación.

Cómo se mide que una voz sintética es buena

Igual que el ASR se mide con WER, el TTS necesita su propia evaluación, y aquí el juez sigue siendo en buena parte humano. La métrica clásica es MOS, Mean Opinion Score: un grupo de oyentes puntúa muestras en una escala de 1 (mala) a 5 (excelente) y se promedia. El procedimiento está estandarizado por la UIT-T en la recomendación P.800.[23]

$$\text{MOS} = \frac{1}{N}\sum_{i=1}^{N} s_i$$

SÍMBOLO	SIGNIFICADO
N	Número de valoraciones recogidas.
s_i	Puntuación de un oyente, de 1 a 5.

MÉTODO	QUÉ EVALÚA	LÍMITE
MOS	Calidad y naturalidad percibida.	Caro, subjetivo, depende del panel y del idioma.
MUSHRA	Comparación fina entre sistemas con referencia oculta y ancla.[24]	Sigue siendo escucha humana.
MOS automático	Modelos que predicen MOS a partir de juicios humanos.	Aproxima, no sustituye, la escucha real.
Inteligibilidad de slots	Si números, fechas y nombres se entienden.	Mide utilidad, no belleza.

Para un agente de voz, la pregunta práctica no es solo “¿suena natural?”. Es “¿se entiende el importe, el identificador y la negación?”. Una voz preciosa que lee “1.250” como “uno punto dos cinco cero” puntúa mal justo donde importa.

WebRTC, WebSocket, RTP y Opus

En navegador, WebRTC es la opción natural para audio bidireccional de baja latencia. RFC 8825 da el marco de protocolos realtime para aplicaciones desplegadas en navegadores.[25] RTP define transporte extremo a extremo para datos de tiempo real como audio y vídeo, aunque no garantiza por sí solo calidad de servicio.[26] W3C mantiene la API WebRTC para navegadores.[27]

OPCIÓN	ÚSALA CUANDO	VIGILA
WebRTC	Navegador/móvil, audio bidireccional, baja latencia, cancelación y NAT traversal.	Señalización, ICE, TURN, permisos, observabilidad.
WebSocket audio	Server-to-server, prototipos controlados, streaming simple.	Jitter, cancelación, codec, backpressure.
HTTP batch	Transcripción de archivos o notas asíncronas.	No sirve para conversación natural.
SIP/telefonía	Call centers y telefonía tradicional.	Integración con centralita, grabación, normativa y latencia.

OpenAI documenta Realtime por WebRTC para escenarios de audio en tiempo real; Azure OpenAI también documenta Realtime vía WebRTC, SIP o WebSocket para enviar audio y recibir audio en tiempo real.[28] La decisión no es de moda. Es de red, cliente, permisos, latencia y operación.

WebRTC por dentro, sin convertir esto en redes avanzadas

MDN resume WebRTC como una pila que incluye ICE, STUN, TURN, SDP y otros protocolos para establecer comunicación realtime entre pares.[29] Para ingeniería de IA, lo importante es saber qué problema resuelve cada pieza:

PIEZA	QUÉ HACE	PREGUNTA DE PRODUCCIÓN
Señalización	Intercambia ofertas, respuestas y configuración inicial por tu backend.	¿Cómo autentico la sesión y cuánto dura el token?
ICE	Busca la mejor ruta de conectividad entre extremos.	¿Qué ocurre en redes corporativas o móviles?
STUN	Ayuda a descubrir la dirección pública detrás de NAT.	¿Funciona fuera de mi WiFi?
TURN	Relé cuando no hay conexión directa.	¿Cuánto coste y latencia añade el relé?
RTP/SRTP	Transporta audio/vídeo en tiempo real, con cifrado en SRTP.	¿Veo pérdida, jitter y bitrate?
RTCP	Informa sobre calidad de transmisión.	¿Uso esas señales en observabilidad?
Data channel	Canal de datos paralelo.	¿Por ahí van eventos, tool calls o metadatos?
Jitter buffer	Suaviza llegada irregular de paquetes.	¿Aumenta latencia para evitar cortes?

El fallo típico de un primer prototipo es que funciona perfecto en localhost y mal en la red de un cliente. No porque “el modelo sea peor”, sino porque la ruta de red cambia: NAT, firewall, TURN, pérdida de paquetes, micrófono Bluetooth, permisos del navegador o región del proveedor.

Herramientas y mercado, con cuidado

Estado de lectura: 14 de junio de 2026.

SERVICIO O FAMILIA	DÓNDE ENCAJA	QUÉ COMPROBARÍA ANTES DE INTEGRARLO
OpenAI Realtime	Voz/audio realtime, eventos de sesión, tools y respuestas de modelo.	Modelo soportado, región, WebRTC/WebSocket, trazas, tool calls, coste de audio y cancelación.
Gemini Live API	Voz y visión en streaming de baja latencia, en preview según documentación.	Estabilidad de preview, límites, vídeo, sesiones, audio input/output y observabilidad.
Azure Speech	Speech-to-text real-time y batch, TTS, traducción y servicios de voz en ecosistema Azure.[30]	Idiomas, custom speech, privacidad, región, logs, SDK y coste.
Amazon Transcribe Streaming	Transcripción en tiempo real de audio entregado como stream.[31]	SDK, protocolo, vocabulario custom, diarización, latencia y pricing por segundo.
Stack propio	ASR/TTS local, WebRTC propio, modelos open weights o servicio interno.	Operación, GPUs/CPU, SLO, mantenimiento, privacidad y calidad por idioma.

Gemini Live documenta gestión de sesiones y reanudación mediante `sessionResumption`, de forma que una desconexión o reset de WebSocket no obliga necesariamente a perder contexto si guardas el token de reanudación.[32] Esto abre una pregunta práctica: si una sesión se reanuda, ¿qué queda guardado, durante cuánto tiempo, con qué permisos y qué se muestra al usuario?

Si vas a producción, no compares solo “calidad de voz”. Compara:

KPI	PREGUNTA
WER por dominio	¿Falla en nombres, códigos, apellidos o tecnicismos?
Latencia p95	¿Qué oye el usuario real, no la demo local?
Barge-in stop p95	¿Se puede interrumpir de verdad?
Tool confirmation accuracy	¿Ejecuta acciones solo con confirmación válida?
Redacción de PII	¿Qué queda en logs y trazas?
Coste por minuto/turno	¿Cuánto cuesta una llamada normal y una mala?
Observabilidad	¿Puedo depurar un incidente?

Tools por voz: no ejecutes una acción porque “pareció decirlo”

En texto, una confirmación puede quedar clara. En voz, hay ASR, ruido, ambigüedad y presión conversacional. Por eso las tools con efecto externo necesitan una política más dura.

TIPO DE TOOL	EJEMPLO	POLÍTICA MÍNIMA
Consulta	“¿Estado de mi solicitud?”	Puede ejecutarse si hay identidad y permisos.
Cálculo	“Calcula el importe pendiente.”	Ejecutar si los datos están disponibles y no cambia estado.
Comunicación	“Envía este correo.”	Confirmación explícita y vista previa.
Datos personales	“Cambia mi teléfono.”	Confirmación, autenticación y canal seguro.
Acción irreversible	“Cancela mi matrícula.”	Revisión humana o doble confirmación fuerte.

Un contrato de tool en voz debería incluir:

```
{
  "tool_name": "cancelar_matricula",
  "effect": "state_change",
  "requires_confirmation": true,
  "confirmation_source": "spoken_transcript",
  "asr_quality_ok": false,
  "allowed": false,
  "reason": "tool destructiva y transcripción sin confirmación explícita"
}
```

La pregunta de ingeniería no es “¿puede el modelo llamar tools?”. Es “¿bajo qué evidencia dejamos que una tool haga algo?”.

Privacidad: la voz trae datos que el usuario no sabe que está entregando

Una conversación puede incluir DNI, teléfono, dirección, voz de terceros, emoción, ruido de fondo y contexto accidental. Incluso si no guardas audio, puedes guardar transcripciones con PII.

Controles mínimos:

CONTROL	QUÉ SIGNIFICA
Consentimiento	El usuario sabe si se graba, transcribe o conserva audio.
Minimización	No guardas audio si solo necesitas decisión y métricas.
Redacción previa	PII se oculta antes de logs, trazas y prompts secundarios.
Retención	Tiempo de conservación explícito por tipo de dato.
Separación	Audio bruto, transcripción, eventos y tools tienen permisos distintos.
Revisión	Casos sensibles van a humano con evidencia mínima necesaria.

Para agentes de voz conectados a RAG, herramientas o CRM, privacidad no es una sección legal al final. Es parte del pipeline.

Arquitectura de producción

Una arquitectura razonable tiene estas capas:

CAPA	RESPONSABILIDAD	SLI QUE MIRARÍA
Cliente de audio	Captura, permisos, WebRTC/WebSocket, reproducción.	audio drop rate, jitter, device errors.
Preprocesado	Resampling, cancelación de eco, ruido, VAD.	false speech, missed speech, endpoint delay.
ASR	Parciales, final, timestamps, idioma, confianza.	WER, slot error rate, finalization latency.
Turn manager	Estado, interrupciones, cierre de turno.	barge-in stop latency, premature cut rate.
Reasoning/tools/ RAG	Respuesta, evidencias, actions.	first token, tool latency, policy blocks.
TTS/playout	Primera voz y reproducción.	first audio latency, audio completion, cancel latency.
Observabilidad	Trazas, métricas, redacción, auditoría.	trace completeness, PII leak rate, replayability.

Un SLO de voz puede sonar así:

SLI	OBJETIVO INTERNO DE EJEMPLO
`first_audio_latency_p95`	Menos de 1300 ms en consultas sin tool.
`barge_in_stop_latency_p95`	Menos de 250 ms desde voz del usuario.
`critical_slot_error_rate`	Menos de 1% en campos críticos evaluados.
`pii_redaction_recall`	100% en patrones obligatorios del dominio.
`tool_confirmation_violation`	0 ejecuciones sin confirmación requerida.

Capacidad: que funcione para una demo no significa que escale

En voz hay una presión que en texto se nota menos: la sesión está viva. Aunque el usuario calle unos segundos, mantienes conexión, estado, buffers, posible contexto y observabilidad.

RTF, Real-Time Factor, ayuda a razonar sobre cómputo:

$$\mathrm{RTF} = \frac{\mathrm{tiempo\_de\_procesamiento}}{\mathrm{duracion\_del\_audio}}$$

RTF	LECTURA
0.25	Procesas cuatro veces más rápido que tiempo real.
1.00	Vas justo a tiempo real.
1.50	No alcanzas: acumulas cola.

Una estimación operativa para dimensionar, no una ley universal:

$$\mathrm{sesiones\_concurrentes\_aprox} = \frac{\mathrm{capacidad\_audio\_segundos\_por\_segundo}}{\mathrm{RTF\_ASR} + \mathrm{RTF\_TTS} + \mathrm{overhead}}$$

La moraleja no es que esa fórmula te dé el número final. La moraleja es que voz no se mide solo por tokens. Mide segundos de audio, conexiones simultáneas, CPU/GPU, TURN, región, TTS, ASR, colas y tools.

Trazas mínimas de una conversación

Si algo falla, una traza útil debería separar spans. No sirve un log gigante con “respuesta generada”.

SPAN	CAMPOS MÍNIMOS
`audio.capture`	dispositivo, sample rate, codec, pérdida, jitter.
`vad.turn_detection`	inicio, fin, silencio, umbral, endpoint delay.
`asr.transcription`	parcial/final, idioma, WER estimada, slots críticos.
`policy.quality_gate`	decisión, flags, umbrales.
`llm.response`	modelo, primer delta, tokens/audio, coste.
`tool.policy_gate`	tool propuesta, permiso, confirmación, decisión.
`tts.synthesis`	voz, primer audio, duración, cancelación.
`privacy.redaction`	tipos redactados, sin valores crudos.

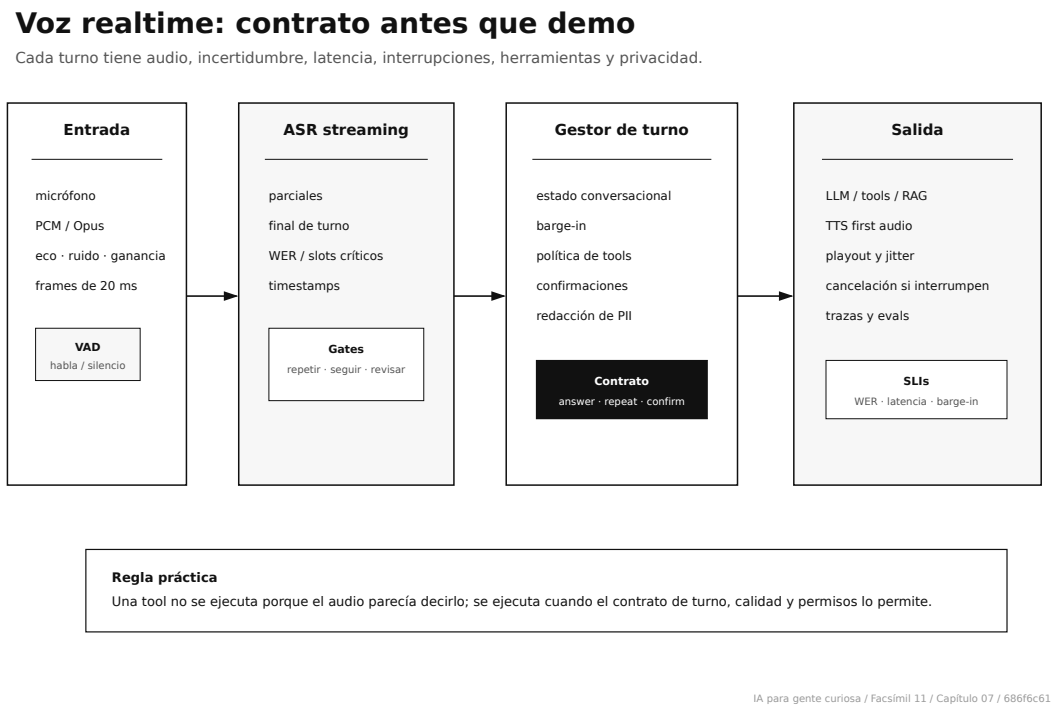
Esta tabla también sirve para revisar proveedores. Si una plataforma no te deja observar estas piezas, quizá sigue siendo válida para prototipo, pero no para un caso con responsabilidad.

Una estimación operativa para el presupuesto de error:

$$\text{ErrorBudget}_{\text{voz}} = 1 - \frac{\text{turnos\_correctos}}{\text{turnos\_totales}}$$

No es una métrica académica universal. Es una forma de obligarte a definir qué cuenta como turno correcto: transcripción aceptable, latencia dentro de SLO, sin PII en claro y sin tool indebida.

Figura: anatomía de una conversación de voz



> La parte difícil de voz no es solo escuchar y hablar: es decidir cuándo una transcripción permite actuar.

Caso práctico: cinco turnos que sí se pueden auditar

El cuaderno del facsímil trae cinco casos:

CASO	QUÉ ENSEÑA	DECISIÓN ESPERADA
Consulta de beca	Voz clara, política y estado operativo.	`answer`.
Ruido de pasillo	WER alto, baja energía y slot crítico mal reconocido: `correo` se vuelve `color`.	`ask_repeat`.
Interrupción	Barge-in mientras el agente habla.	`stop_and_answer`.
Datos sensibles	DNI y teléfono dictados por voz.	`answer` con redacción previa.
Cancelación de matrícula	Tool destructiva sin confirmación.	`confirm_before_tool`.

La práctica no pretende simular un ASR real con un modelo enorme. Pretende algo más útil para aprender ingeniería: que puedas cambiar umbrales, ver decisiones, medir slots críticos y defender por qué un turno se automatiza o no.

En el día a día

La voz aparece cuando alguien quiere «hablar con el asistente» y descubre que un agente de voz no es un chatbot con micrófono. El día a día se llena de problemas que no existen en texto: el ASR oye «color» donde el usuario dijo «correo», una pausa para respirar corta el turno antes de tiempo, el usuario interrumpe y el sistema sigue recitando, o una transcripción dudosa intenta disparar una acción irreversible. La latencia deja de ser una métrica abstracta: si el sistema tarda en responder, el usuario interrumpe; si no entiende el barge-in, se enfada.

El segundo frente es que el trabajo no va de «qué voz suena mejor», sino de contratos de turno: cuándo empieza y termina la voz, qué transcripción es estable, qué slot crítico hay que confirmar, cuándo se redacta un dato personal y cuándo una herramienta necesita aprobación. El equipo que mide WER, latencia de primera voz, barge-in y error en slots críticos es el que sabe cuándo su agente de voz es operable y cuándo solo es una demo simpática.

Y hay un tercer frente que se olvida hasta que estalla: la privacidad de la voz. Una llamada trae mucho más de lo que el usuario cree que entrega (acento, entorno, otra persona de fondo, tono emocional, posibles datos biométricos), y conservar audio bruto durante meses rara vez está justificado. El equipo que no lo piensa desde el principio acaba con grabaciones sensibles en un almacén sin política de retención, y descubre el problema el día que alguien pregunta qué guardan, durante cuánto y con qué permiso.

Por qué debería importarte

Entender la voz como un sistema de tiempo real cambia qué mides y qué proteges. Te lleva a no ejecutar una herramienta porque «el audio pareció decirlo», a medir la transcripción por separado antes de juzgar el resumen, y a tratar la latencia y el barge-in como requisitos, no como detalles de experiencia de usuario. El caso que mejor lo enseña es el del nombre o el importe: si el ASR transcribe mal «mil» como «cien», un resumen impecablemente redactado puede ser falso, y el modelo final no se equivocó, solo repitió con buena prosa un error que ya venía hecho. Por eso un agente de voz serio pide confirmación en los slots críticos, redacta la PII antes de los logs y deja una traza que se puede reproducir. En voz, los fallos se notan más y se perdonan menos.

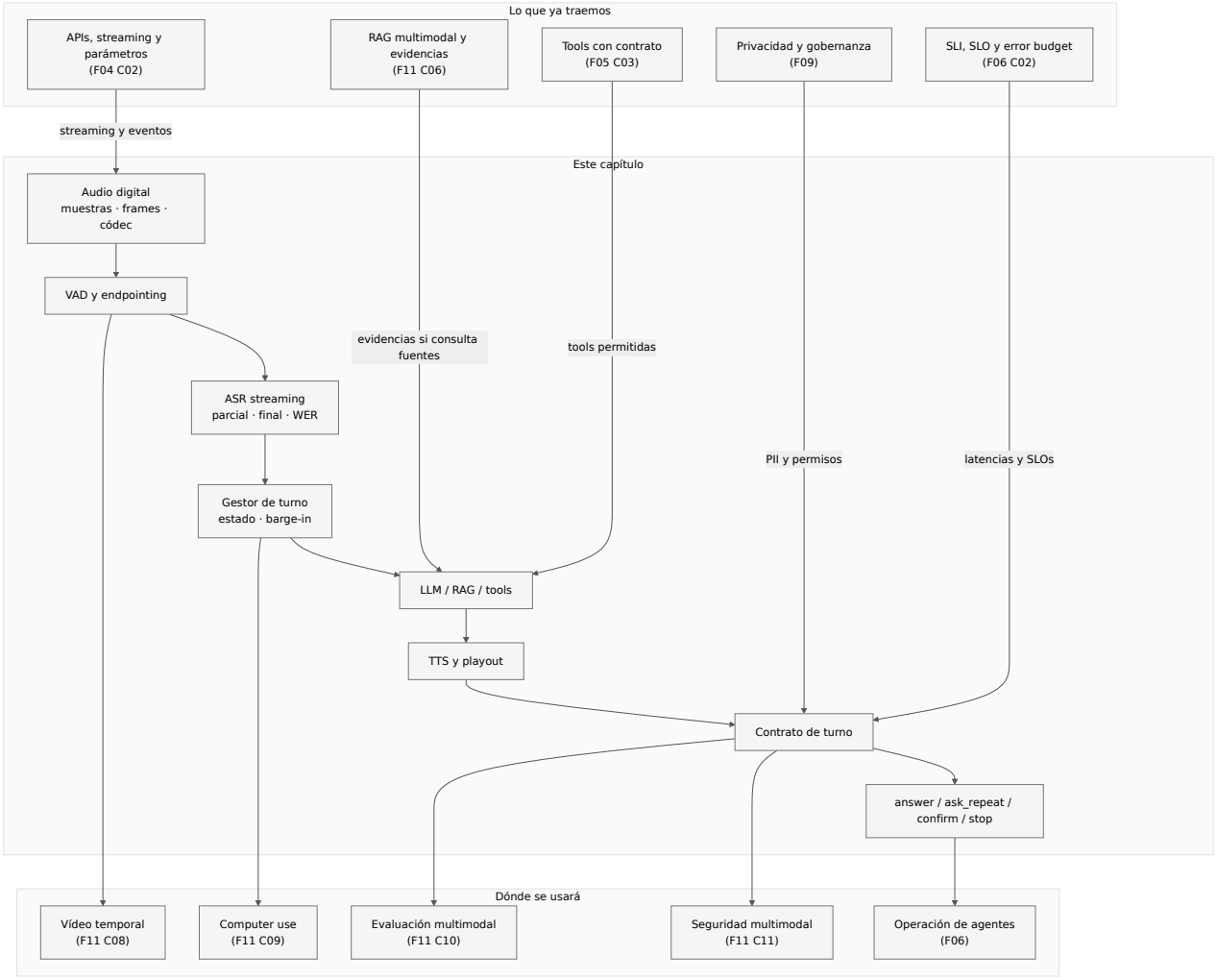
Dónde volverá a aparecer

CAPÍTULO FUTURO	QUÉ REUTILIZA
Capítulo 08	Vídeo añade frames y eventos temporales a la misma idea de stream.
Capítulo 09	Computer use necesita turnos, permisos y cancelación antes de actuar.
Capítulo 10	Evaluaremos multimodalidad con métricas por modalidad y casos de fallo.
Capítulo 11	Privacidad y seguridad de audio entran en operación real.
Fascículo 06	SLOs, trazas, runbooks y gates de release para sistemas en producción.

Dónde solía tropezar yo

TROPIEZO	POR QUÉ ES UN PROBLEMA	ANTÍDOTO
Pensar que ASR equivale a intención	La transcripción puede ser incorrecta justo en la palabra crítica.	Separa ASR, intención, permiso y tool.
Medir solo WER medio	Una WER baja puede esconder error en un slot peligroso.	Evalúa slots críticos y acciones.
Cerrar turno demasiado rápido	El usuario hace una pausa normal y el sistema responde antes de tiempo.	Mide endpointing y cortes prematuros.
No implementar barge-in real	El agente se siente como una locución imposible de parar.	Cancela TTS y generación anterior.
Guardar trazas con PII	La voz suele traer datos personales en claro.	Redacta antes de logs y minimiza retención.
Permitir tools por una frase ambigua	“Cancela si...” no es confirmación robusta.	Usa confirmación explícita y revisión.
Comparar proveedores por demo	La demo no muestra acentos, ruido, coste ni SLO.	Evalúa con tus audios, tus tools y tus usuarios.

Cómo encaja todo



Vocabulario aprendido

TÉRMINO	DEFINICIÓN
Audio digital	Señal sonora convertida en muestras discretas.
Frame	Pequeño bloque de audio usado para procesar en streaming.
ASR	Sistema que convierte audio hablado en texto.
TTS	Sistema que convierte texto en voz.
VAD	Detector de actividad de voz.
Endpointing	Decisión de cerrar un turno de usuario.
Barge-in	Interrupción del usuario mientras el agente está hablando.
WER	Word Error Rate: errores de transcripción por palabra.
Slot crítico	Campo cuyo error cambia la decisión, la tool o el riesgo.
Diarización	Separar quién habla y cuándo en un audio con varios hablantes.
RTF	Relación entre tiempo de procesamiento y duración del audio.
Opus	Códec de audio interactivo para voz y baja latencia.
WebRTC	Stack para comunicación multimedia realtime en navegador y clientes.
ICE/STUN/TURN	Piezas de conectividad para atravesar NAT, descubrir rutas o usar relés.
Contrato de turno	Registro estructurado de audio, transcripción, decisión, métricas y permisos.

Antes de pasar página

Hazte estas preguntas:

1. ¿Qué formato de audio entra en mi sistema?
2. ¿Tengo transcripción parcial y final?
3. ¿Qué WER o error de slot tolero antes de pedir repetición?
4. ¿Cuánto espero antes de cerrar un turno?

5. ¿Se puede interrumpir al agente mientras habla?
6. ¿Qué tools no se ejecutan jamás sin confirmación explícita?
7. ¿Dónde redacto PII: antes o después de logs?
8. ¿Tengo un dataset con ruido, acentos, pausas, negaciones y PII?
9. ¿Sé qué ocurre si la conexión WebRTC cae y se reanuda?
10. ¿Puedo reproducir una conversación problemática sin exponer datos sensibles?

Si no puedes contestar, todavía no tienes un agente de voz operativo. Tienes una demo.

En resumen

IDEA	QUÉ DEBERÍAS LLEVARTE
La voz es tiempo real.	No basta con transcribir bien; hay que medir latencia, turnos e interrupciones.
ASR produce hipótesis.	Una transcripción no es permiso para actuar.
El contrato de turno es la pieza operativa.	Une audio, texto, slots críticos, decisión, tools, privacidad y métricas.
Barge-in cambia la experiencia.	Si el usuario no puede interrumpir, no hay conversación natural.
WebRTC también es ingeniería.	La calidad depende de ICE, TURN, jitter, región, codec y observabilidad.
La práctica debe ser auditable.	Cada turno descargable debe poder ejecutarse, medirse y defenderse.

Para saber más

- Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C. y Sutskever, I. (2023). *Robust Speech Recognition via Large-Scale Weak Supervision*. Proceedings of ICML 2023. https://proceedings.mlr.press/v202/radford23a.html
- Koenecke, A., Choi, A. S. G., Mei, K., Schellmann, H. y Sloane, M. (2024). *Careless Whisper: Speech-to-Text Hallucination Harms*. ACM FAccT 2024. https://doi.org/10.1145/3630106.3658996
- Zeghidour, N., Luebs, A., Omran, A., Skoglund, J. y Tagliasacchi, M. (2021). *SoundStream: An End-to-End Neural Audio Codec*. IEEE/ACM TASLP. https://arxiv.org/abs/2107.03312

- Défossez, A., Copet, J., Synnaeve, G. y Adi, Y. (2022). *High Fidelity Neural Audio Compression* (EnCodec). https://arxiv.org/abs/2210.13438
- Baevski, A., Zhou, H., Mohamed, A. y Auli, M. (2020). *wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations*. NeurIPS. https://arxiv.org/abs/2006.11477
- Gulati, A. y otros (2020). *Conformer: Convolution-augmented Transformer for Speech Recognition*. https://arxiv.org/abs/2005.08100
- Graves, A. (2012). *Sequence Transduction with Recurrent Neural Networks*. https://arxiv.org/abs/1211.3711
- IETF. (2012). *RFC 6716: Definition of the Opus Audio Codec*. https://datatracker.ietf.org/doc/html/rfc6716
- Schulzrinne, H., Casner, S., Frederick, R. y Jacobson, V. (2003). *RFC 3550: RTP: A Transport Protocol for Real-Time Applications*. https://datatracker.ietf.org/doc/html/rfc3550
- OpenAI. (2026). *Realtime and audio guide*. https://developers.openai.com/api/docs/guides/realtime
- OpenAI. (2026). *Realtime conversations*. https://developers.openai.com/api/docs/guides/realtime-conversations
- Google AI for Developers. (2026). *Gemini Live API overview*. https://ai.google.dev/gemini-api/docs/live-api
- Panayotov, V., Chen, G., Povey, D. y Khudanpur, S. (2015). *LibriSpeech: An ASR Corpus Based on Public Domain Audio Books*. https://www.openslr.org/12
- Ardila, R. y otros (2020). *Common Voice: A Massively-Multilingual Speech Corpus*. https://aclanthology.org/2020.lrec-1.520/
- Bredin, H. y otros (2019). *pyannote.audio: neural building blocks for speaker diarization*. https://arxiv.org/abs/1911.01255
- NIST. (2026). *SCTK / sclite documentation*. https://github.com/usnistgov/SCTK/blob/master/doc/sclite.htm

---

Notas

1. Shannon, C. E. (1949). Communication in the Presence of Noise. *Proceedings of the IRE*, 37(1), 10-21. https://doi.org/10.1109/JRPROC.1949.232969. ↩

2. IETF. (2012). *RFC 6716: Definition of the Opus Audio Codec*. https://datatracker.ietf.org/doc/html/rfc6716. Consultado el 14 de junio de 2026. ↩

3. OpenAI. (2026). *Realtime and audio guide*. https://developers.openai.com/api/docs/guides/realtime. Consultado el 14 de junio de 2026. ↩

4. Google AI for Developers. (2026). *Gemini Live API overview*. https://ai.google.dev/gemini-api/docs/live-api. Consultado el 14 de junio de 2026. ↩

5. Zeghidour, N., Luebs, A., Omran, A., Skoglund, J. y Tagliasacchi, M. (2021). *SoundStream: An End-to-End Neural Audio Codec*. IEEE/ACM Transactions on Audio, Speech, and Language Processing, 30, 495-507. https://arxiv.org/abs/2107.03312. ↩

6. Défossez, A., Copet, J., Synnaeve, G. y Adi, Y. (2022). *High Fidelity Neural Audio Compression*. https://arxiv.org/abs/2210.13438. ↩

7. OpenAI. (2026). *Realtime conversations*. https://developers.openai.com/api/docs/guides/realtime-conversations. Consultado el 14 de junio de 2026. ↩

8. OpenAI. (2026). *Realtime server events reference*. https://developers.openai.com/api/reference/resources/realtime/server-events/. Consultado el 14 de junio de 2026. ↩

9. Graves, A., Fernández, S., Gomez, F. y Schmidhuber, J. (2006). *Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks*. Proceedings of ICML 2006, 369-376. https://doi.org/10.1145/1143844.1143891. ↩

10. Graves, A. (2012). *Sequence Transduction with Recurrent Neural Networks*. https://arxiv.org/abs/1211.3711. ↩

11. Baevski, A., Zhou, H., Mohamed, A. y Auli, M. (2020). *wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations*. NeurIPS. https://arxiv.org/abs/2006.11477. ↩

12. Gulati, A. y otros (2020). *Conformer: Convolution-augmented Transformer for Speech Recognition*. https://arxiv.org/abs/2005.08100. ↩

13. Radford, A. y otros (2023). *Robust Speech Recognition via Large-Scale Weak Supervision*. Proceedings of ICML 2023, 28492-28518. https://proceedings.mlr.press/v202/radford23a.html. ↩

14. Koenecke, A., Choi, A. S. G., Mei, K., Schellmann, H. y Sloane, M. (2024). *Careless Whisper: Speech-to-Text Hallucination Harms*. Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency (FAccT '24), 1672-1681. https://doi.org/10.1145/3630106.3658996. ↩

15. NIST. (2026). *SCTK / sclite documentation*. https://github.com/usnistgov/SCTK/blob/master/doc/sclite.htm. Consultado el 14 de junio de 2026. ↩

16. Bredin, H. y otros (2019). *pyannote.audio: neural building blocks for speaker diarization*. https://arxiv.org/abs/1911.01255. ↩

17. Panayotov, V., Chen, G., Povey, D. y Khudanpur, S. (2015). *LibriSpeech: An ASR Corpus Based on Public Domain Audio Books*. ICASSP. https://www.openslr.org/12. ↩

18. Ardila, R. y otros (2020). *Common Voice: A Massively-Multilingual Speech Corpus*. LREC. https://aclanthology.org/2020.lrec-1.520/ ↩

19. van den Oord, A. y otros (2016). *WaveNet: A Generative Model for Raw Audio*. https://arxiv.org/abs/1609.03499. ↩

20. Shen, J. y otros (2018). *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions*. ICASSP. https://arxiv.org/abs/1712.05884. ↩

21. Ren, Y. y otros (2019). *FastSpeech: Fast, Robust and Controllable Text to Speech*. NeurIPS. https://arxiv.org/abs/1905.09263. ↩

22. Kong, J., Kim, J. y Bae, J. (2020). *HiFi-GAN: Generative Adversarial Networks for Efficient and High Fidelity Speech Synthesis*. NeurIPS. https://arxiv.org/abs/2010.05646. ↩

23. ITU-T. (1996). *Recommendation P.800: Methods for Subjective Determination of Transmission Quality*. https://www.itu.int/rec/T-REC-P.800. ↩

24. ITU-R. (2015). *Recommendation BS.1534: Method for the Subjective Assessment of Intermediate Quality Level of Audio Systems (MUSHRA)*. https://www.itu.int/rec/R-REC-BS.1534. ↩

25. Alvestrand, H. (2021). *RFC 8825: Overview: Real-Time Protocols for Browser-Based Applications*. https://datatracker.ietf.org/doc/html/rfc8825. Consultado el 14 de junio de 2026. ↩

26. Schulzrinne, H., Casner, S., Frederick, R. y Jacobson, V. (2003). *RFC 3550: RTP: A Transport Protocol for Real-Time Applications*. https://datatracker.ietf.org/doc/html/rfc3550. Consultado el 14 de junio de 2026. ↩

27. W3C. (2026). *WebRTC: Real-Time Communication in Browsers*. https://www.w3.org/TR/webrtc/. Consultado el 14 de junio de 2026. ↩

28. Microsoft. (2026). *Use the GPT Realtime API via WebRTC*. https://learn.microsoft.com/en-us/azure/foundry/openai/how-to/realtime-audio-webrtc. Consultado el 14 de junio de 2026. ↩

29. MDN Web Docs. (2026). *Introduction to WebRTC protocols*. https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Protocols. Consultado el 14 de junio de 2026. ↩

30. Microsoft. (2026). *Speech-to-text documentation*. https://learn.microsoft.com/en-us/azure/ai-services/speech-service/index-speech-to-text. Consultado el 14 de junio de 2026. ↩

31. AWS. (2026). *Transcribing streaming audio*. https://docs.aws.amazon.com/transcribe/latest/dg/streaming.html. Consultado el 14 de junio de 2026. ↩

32. Google AI for Developers. (2026). *Session management with Live API*. https://ai.google.dev/gemini-api/docs/live-api/session-management. Consultado el 14 de junio de 2026. ↩

3. Koenecke, A., Choi, A. S. G., Mei, K., Schellmann, H. y Sloane, M. (2024). *Careless Whisper: Speech-to-Text Hallucination Harms*. ACM FAccT 2024. https://doi.org/10.1145/3630106.3658996. ↩