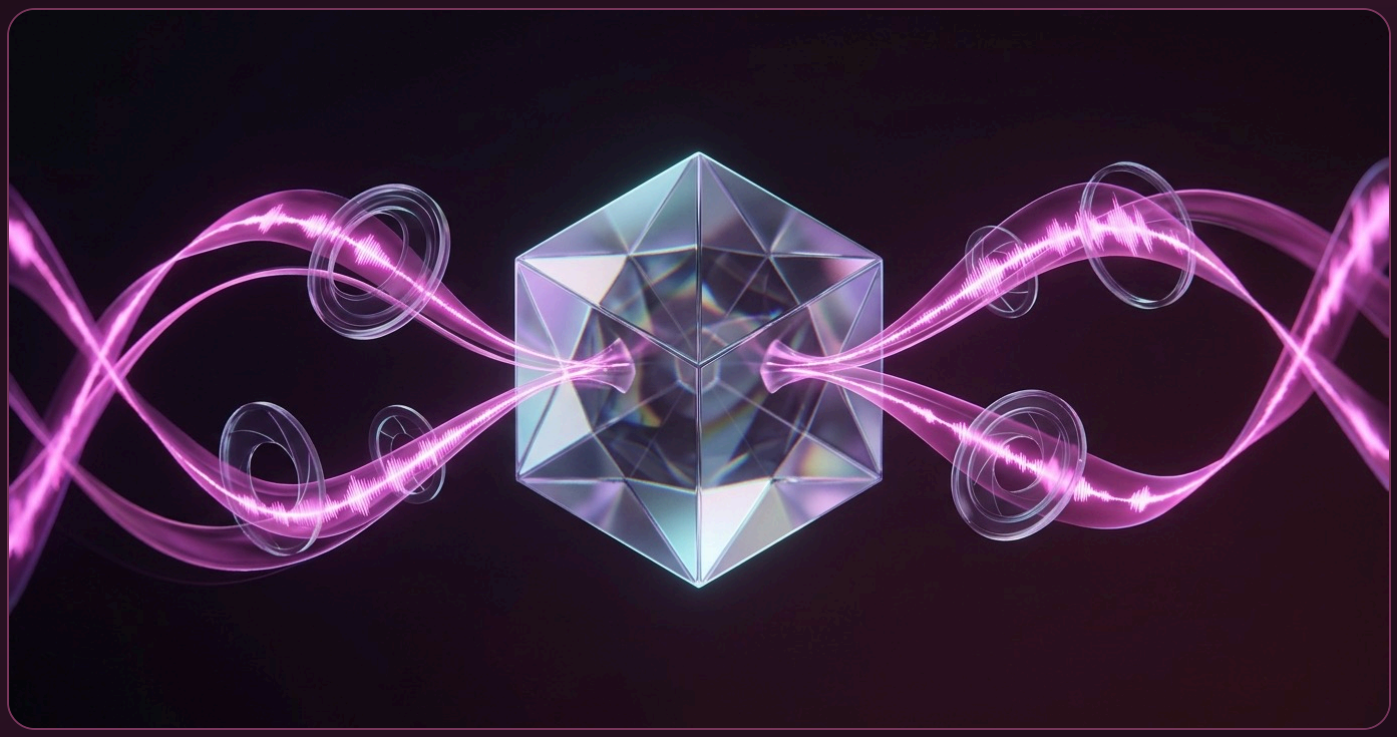
OPENAI CODEX, AGENTS & REALTIME STUDIOS • FIELD GUIDE

OpenAI Realtime Voice & Vision

Build natural, sub-second speech-to-speech and live camera applications with `gpt-realtime-2.1`—mastering ephemeral WebRTC auth, SIP telephony bridges, `addImage()` visual context, instant VAD barge-in, and non-blocking async MCP tools.

`gpt-realtime-2.1` | WebRTC Ephemeral Auth | SIP / PSTN Bridge | `addImage()` Vision | VAD Barge-In | Async MCP Calls



Native Speech-to-Speech
Stream audio directly in and out at ~250ms latency while preserving tone, laughter, hesitation, and natural cadence.

Live Camera & Screen Vision
Inject camera or screen frames mid-conversation with `addImage()` so the assistant sees what you see without stalling audio.

Full-Duplex & Async Tools
Interrupt cleanly with `interrupt()` and run slow CRM or DB lookups off the audio thread via `tool()` & `hostedMcpTool()`.

Single-Model Speech-to-Speech & Live Vision

Traditional voice bots chain three separate models together—speech-to-text, a text LLM, and text-to-speech—introducing awkward 1.5–3 second pauses and erasing vocal emotion. OpenAI's `gpt-realtime-2.1` processes audio and visual frames inside one native multimodal stream.

Quick Navigation

1. Single-Model Architecture & Event Loop p. 2
2. Models & Transport Decision Matrix p. 3
3. Annotated Session Inspector UI Tour p. 4
4. Ephemeral WebRTC, WebSockets & SIP p. 5
5. `addImage()` Vision & Voice Personas p. 6
6. Server VAD, Barge-In & Async MCP p. 7
7. Production Audio Hygiene & Guardrails p. 8
8. 5 Real-World Copy-Paste Recipes p. 9
9. Latency Budget, Telemetry & Recovery p. 10
10. Cross-Tool Synergy Pipeline p. 11
11. Troubleshooting Clinic & Pre-Flight p. 12
Golden Rule: Never Ship Static Keys p. 5

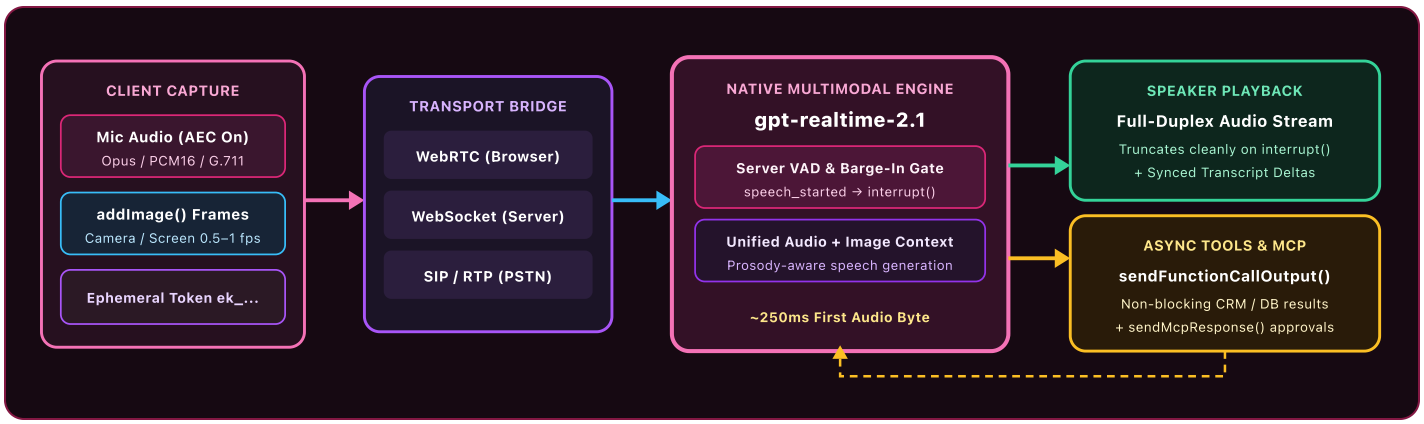
Legacy Chained Pipeline (1.5s–3.0s)

Audio → **STT transcription** (loses sighs, urgency, accents) → **Text LLM** (waits for full transcript) → **TTS synthesizer** (reads Markdown stiffly). Interruptions feel clumsy because three independent queues must be flushed in sequence.

Native Realtime Stream (~250ms)

Audio + `addImage()` frames flow directly into `gpt-realtime-2.1` over WebRTC, WebSocket, or SIP. The model hears tone, tracks turn boundaries natively, and streams spoken audio back while running background MCP tools.

Full-Duplex Transport & Multimodal Event Loop



Why Full-Duplex Streaming Feels Effortlessly Human

Because audio input and audio output travel simultaneously without waiting for batch transcription, users can interject mid-sentence with "Wait, skip step two" and the model immediately stops speaking, adapts to their tone, and pivots in real time.

Model Tiers & Transport Selection Matrix

Every Realtime build begins with two architectural choices: which native speech-to-speech model fits your reasoning and budget profile, and which network transport carries your audio frames cleanly from the user's device to OpenAI.

1 `gpt-realtime-2.1` (Flagship)

- **Best For:** Complex troubleshooting, live camera inspection, multi-tool MCP orchestration, and empathetic customer care.
- **Strengths:** Deepest instruction adherence, nuanced vocal prosody, reliable structured tool schemas, and sharp visual detail on dense UIs or hardware.
- **Tradeoff:** Higher per-minute audio and image token rate; reserve for high-value interactive sessions.

2 `gpt-realtime-2.1-mini` (High-Volume)

- **Best For:** Receptionist routing, appointment reminders, voice search, and high-concurrency kiosks or outbound campaigns.
- **Strengths:** Dramatically lower per-minute spend, crisp snappy turn-taking, and low first-byte latency on straightforward intents.
- **Tradeoff:** Lighter multi-step reasoning; keep prompts concise and hand off complex edge cases when needed.

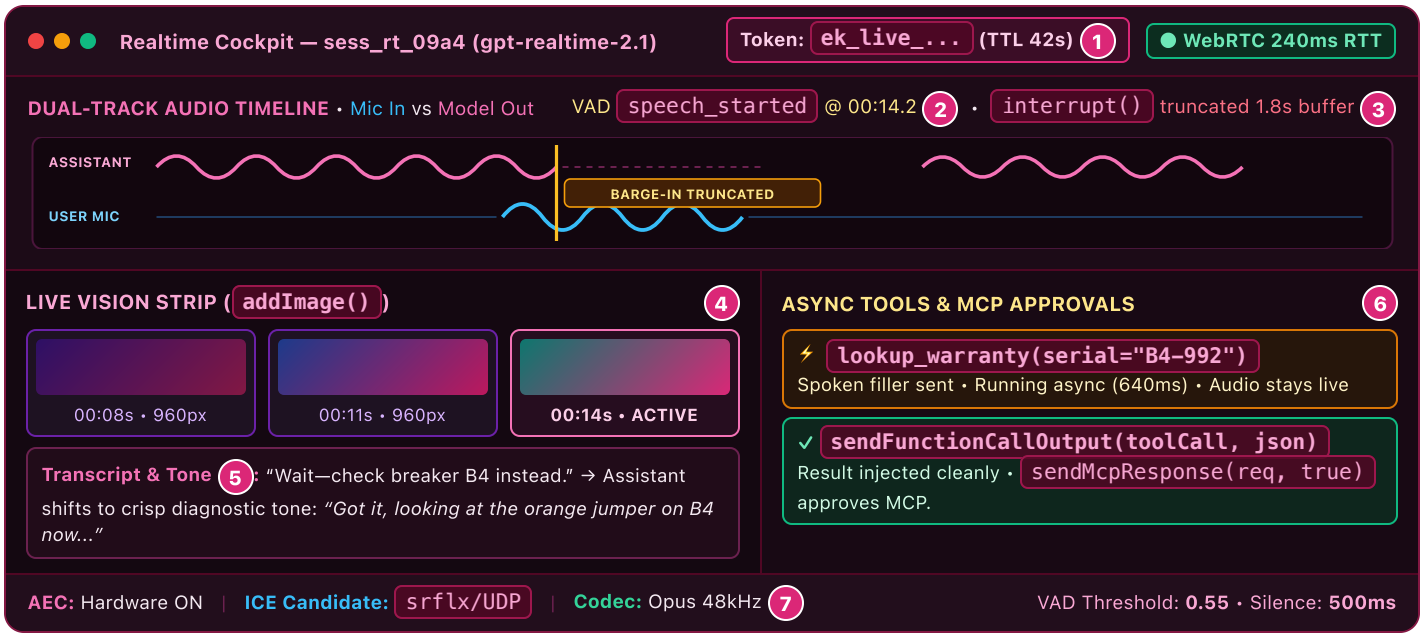
Transport Decision Matrix: WebRTC vs. WebSockets vs. SIP / RTP

Dimension	WebRTC (`@openai/agents/realtime`)	WebSockets (Server Relay)	SIP / RTP Telephony (PSTN)
Where It Runs	**Direct from browser or mobile app** straight to OpenAI edge media servers.	**Backend-to-OpenAI** (Node.js, Python, or edge workers relaying audio buffers).	**Carrier trunks** (Twilio, Telnyx, Vonage) bridging traditional phone numbers.
Authentication	Short-lived **ephemeral client secret** (`ek_...`) minted on your server.	Standard server-side `OPENAI_API_KEY` held securely in backend env vars.	Trunk SIP credentials + webhook signature verification on your call controller.
Audio & Latency	**Lowest latency (~200–280ms);** UDP media with built-in jitter buffer & Opus 48kHz.	Slightly higher (adds server hop); uses raw `PCM16` 24kHz or G.711 frames over TCP.	Carrier-grade voice (~300–450ms); uses `G.711 mu-law / a-law` at 8kHz.
Live Vision (`addImage()`)	**Ideal:** Send camera or `getDisplayMedia()` screen frames over the data channel.	Supported via base64 image events relayed through your WebSocket server.	Voice-only on standard PSTN calls (pair with SMS link if visual proof is needed).
Best Use Cases	Web copilots, visual field support, browser tutors, React/Next.js voice apps.	Custom DSP pipelines, compliance recording proxies, server-hosted bots.	Inbound support hotlines, IVR replacement, booking desks, dispatch phones.

Rule of Thumb: Never Proxy Browser Audio Through Your Own WebSocket If WebRTC Fits
Proxying microphone chunks from a browser through your Node or Python backend over TCP WebSockets adds head-of-line blocking and extra round-trip jitter. Let browsers talk directly to OpenAI over **WebRTC** using an ephemeral token, and keep your server solely for minting secrets and running privileged MCP tools.

Annotated Tour: Realtime Session Inspector

Unlike stateless text chat, a Realtime session coordinates audio waveforms, camera frames, voice activity boundaries, and background tool calls simultaneously. Here are the seven live telemetry points inside a healthy session.



Interface Legend: The 7 Live Realtime Control Points

1. **Ephemeral Token Badge:** Shows the short-lived `ek_...` client secret minted by your backend—valid just long enough to complete the WebRTC SDP handshake.
2. **Server VAD Markers:** Pinpoints exact millisecond boundaries where OpenAI's voice activity detector detects `speech_started` and `speech_stopped`.
3. `interrupt()` **Truncation Gate:** Fires the moment user speech overlaps assistant playback, clearing buffered audio so the model never talks over the caller.
4. `addImage()` **Frame Strip:** Displays throttled camera or screen snapshots pushed silently into conversation state without interrupting audio flow.
5. **Synced Transcript & Prosody:** Streams dual transcripts for captions and audit logs while preserving vocal inflection inside the raw audio turn.
6. **Async Tools &** `sendMcpResponse()`**:** Runs slow CRM/SQL lookups out-of-band via async `tool()` / `sendFunctionCallOutput()`, while `sendMcpResponse(req, true)` approves MCP calls.
7. **Audio Hygiene & ICE Health Strip:** Verifies browser echo cancellation (AEC), UDP NAT traversal, and matching sample rates (Opus 48kHz / PCM16 24kHz / G.711 8kHz) so self-echo never causes false interruptions.

WebRTC Ephemeral Secrets, WebSockets & SIP

Golden Rule: Never embed your permanent `OPENAI_API_KEY` in browser JavaScript or mobile bundles. Instead, mint a single-use ephemeral key on your backend via `POST /v1/realtime/client_secrets` and hand only that short-lived token to the client.

1. Backend Route: Mint Ephemeral Secret (Node / Next.js)

```
// POST /api/realtime-token (runs strictly on your server)
const resp = await
fetch('https://api.openai.com/v1/realtime/client_secrets', {
  method: 'POST',
  headers: {
    'Authorization': 'Bearer ' + process.env.OPENAI_API_KEY,
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    expires_after: { anchor: 'created_at', seconds: 60 },
    session: { type: 'realtime', model: 'gpt-realtime-2.1' }
  })
});
// Default TTL is 600s if expires_after is omitted
const { value: ephemeralKey } = await resp.json();
```

2. Browser Client: Connect Over WebRTC

```
import { RealtimeAgent, RealtimeSession } from
'@openai/agents/realtime';

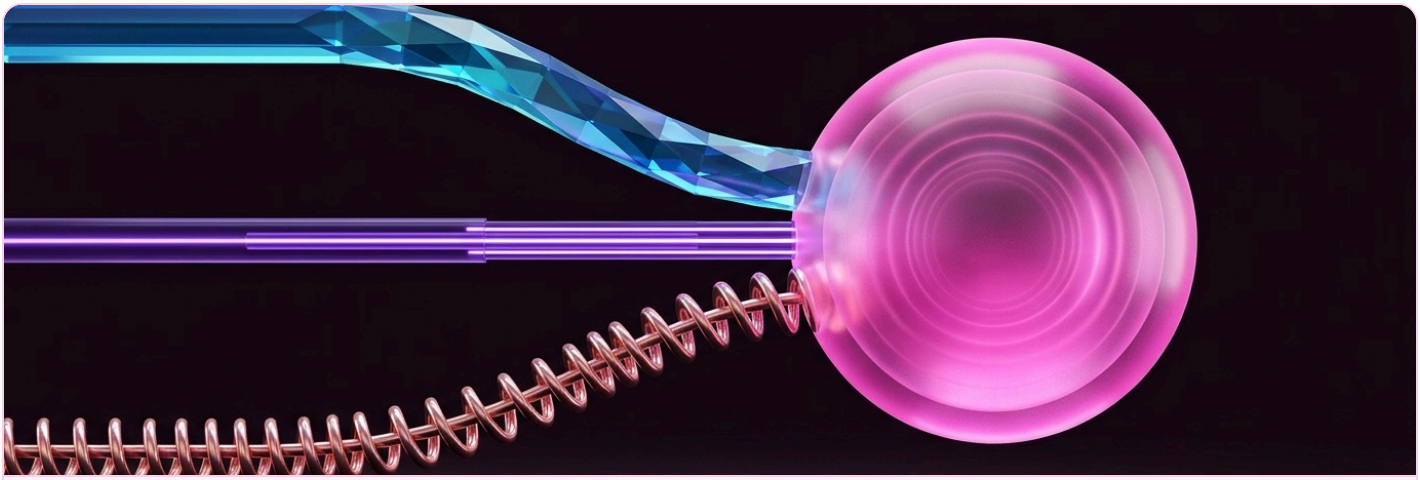
const agent = new RealtimeAgent({
  name: 'SupportCopilot',
  instructions: 'Speak concisely in warm plain sentences.'
});
const session = new RealtimeSession(agent);
const { ephemeralKey } = await fetch('/api/realtime-token',
{ method: 'POST' }).then(r => r.json());
// Connects directly over WebRTC in browsers
await session.connect({ apiKey: ephemeralKey });
```

A WebSockets for Server Relays

When your backend already owns an audio stream (such as a Twilio Media Stream, game server, or compliance proxy), connect directly from Node or Python to `wss://api.openai.com/v1/realtime?model=gpt-realtime-2.1` using your server-side API key and exchange base64 audio events.

B SIP / RTP Telephony Bridges

Connect Twilio, Telnyx, or enterprise SBC trunks via SIP signaling or carrier WebSocket bridges. Match codec settings to `g711_ulaw` (North America/Japan) or `g711_alaw` (Europe/global) at 8kHz so phone calls sound crisp with zero transcoding artifacts.



Three Transports, One Conversation Engine: Whether a caller joins via browser **WebRTC** (sapphire arch), a server-hosted **WebSocket** relay (violet fiber), or a PSTN **SIP trunk** (rose-gold coil), every pathway converges into the same `gpt-realtime-2.1` full-duplex session.

Live Vision via `addImage()` & Voice Personas

Unlike text models where you upload an image and wait for a turn, `addImage()` lets your app push camera or screen frames silently into an ongoing voice conversation—giving the assistant shared eyes without interrupting audio.

```
// Capture a clean 960px JPEG frame every 2 seconds (or on scene change) and push silently:
const canvas = document.createElement('canvas');
canvas.width = 960; canvas.height = 540;
canvas.getContext('2d')?.drawImage(videoEl, 0, 0, 960, 540);
const frameDataUrl = canvas.toDataURL('image/jpeg', 0.72);

// triggerResponse: false stores the visual frame in context without forcing the model to speak immediately
await session.addImage(frameDataUrl, { triggerResponse: false });
```

Throttle to 0.5–1 fps

Sending 30 frames per second wastes tokens and saturates mobile uplinks. One crisp frame every 1–2 seconds (or on button tap) covers 95% of visual tasks.

Cap Width at 768–1024px

Downscale 4K webcam or Retina screen grabs to `960px` JPEG at `0.72` quality before calling `addImage()` to keep payloads fast.

Silent Context Updates

Keep `triggerResponse: false` for background frames so the model only comments when the user asks or a safety hazard appears.

Voice & Vision System Prompt Anatomy (5 Color-Coded Clauses)

[1. Voice & Register] [2. Pacing & Cadence] [3. Turn-Taking Politeness] [4. Spoken Brevity Rules] [5. Visual Grounding]

"You are Maya, a senior HVAC field coach. Speak in a warm, steady, reassuring tone like a calm coworker standing beside the technician. Use a crisp conversational pace and pause naturally after giving one physical step. If the caller interrupts mid-sentence, stop immediately, acknowledge their question in four words or fewer, and answer it directly. Keep every reply under two short sentences, use plain spoken words, and never read out bullet lists, asterisks, URLs, or Markdown symbols. Reference only components clearly visible in the latest camera frame; if wiring labels are blurry, ask the caller to move the camera closer instead of guessing."

Write Prompts for Ears, Not Eyes

Text LLMs love numbered lists, bold headers, and nested parentheses. Out loud, those sound robotic and inflate turn duration. Explicitly ban Markdown formatting and instruct the model to give **one concrete step at a time**, waiting for the caller's "Okay, done" before moving on.

Native VAD, Barge-In & Async MCP Calls

Natural conversation demands two hard engineering guarantees: when the user cuts in, buffered assistant audio must stop within milliseconds; and when the model runs a slow database query, the voice channel must never go dead silent.

1. Tuning Server VAD & Clean `interrupt()`

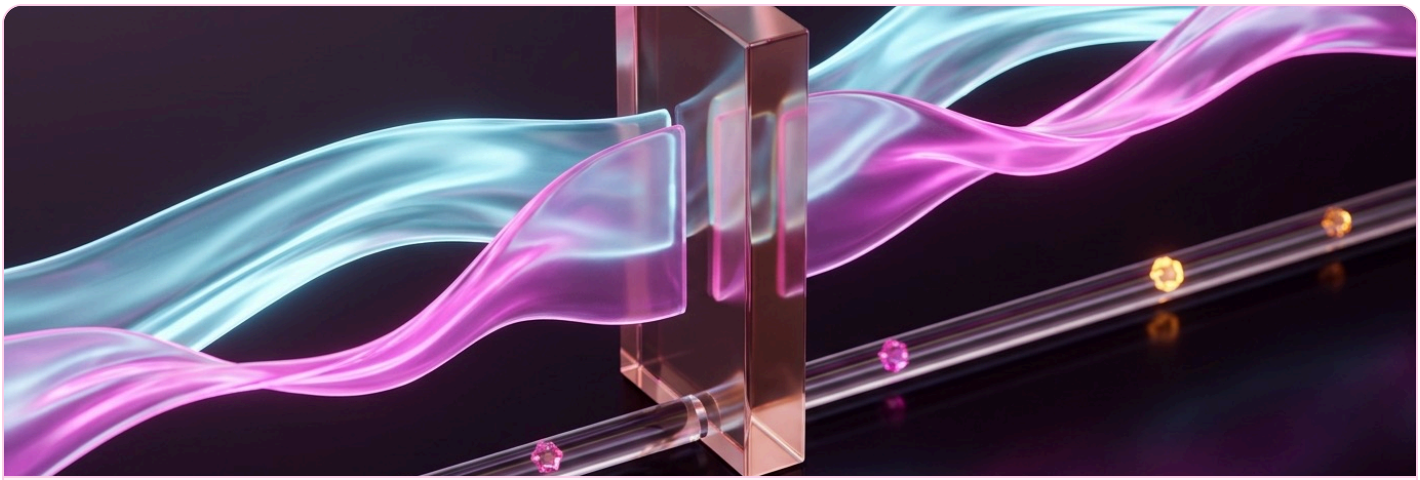
```
// Configure Server VAD under audio.input
config: { audio: { input: { turnDetection: {
  type: 'server_vad', threshold: 0.55,
  prefix_padding_ms: 300, silence_duration_ms: 500
}}}}
// Flush client speaker queue on VAD interrupt:
session.on('audio_interrupted', () =>
player.stopAndFlush());
// Or call session.interrupt() on a Stop button
```

2. Async Tools & Hosted MCP Approvals

```
// Run slow DB/API work inside async tool() or out-of-band:
const order = await db.getOrder(toolCall.callId);
session.transport.sendFunctionCallOutput(
  toolCall, JSON.stringify(order), true
);
// For hosted MCP tools needing human approval:
session.on('mcp_tool_call_approval_request', (req) => {
  session.transport.sendMcpResponse(req, true);
});
```

Why Calling `interrupt()` Keeps Memory Accurate

When a listener barges in two seconds into a ten-second audio reply, the model may have already generated text tokens ahead of the speaker buffer. Calling `interrupt()` (which triggers `conversation.item.truncate`) tells OpenAI the exact millisecond the listener actually heard, preventing the model from hallucinating that it finished explaining steps the user never heard.



Two Parallel Planes of Responsiveness: Top lane—incoming and outgoing soundwaves part instantaneously at the **VAD Barge-In Gate** on `audio_interrupted` / `interrupt()`. Bottom lane—glowing **Async Tool & MCP Sparks** return data via `sendFunctionCallOutput()` and approve gated MCP calls via `sendMcpResponse()` with zero dead air.

Production Audio Hygiene & Guardrails

Ninety percent of “the AI interrupted itself” or “the phone audio sounded robotic” bug reports are acoustic and codec hygiene issues—not model defects. Lock down these four production layers before going live.

1 Acoustic Echo Cancellation (AEC)

Without AEC, your laptop or phone speaker audio feeds straight back into the microphone—causing the assistant to hear its own voice and trigger a false barge-in loop.

- In browsers, always request `echoCancellation: true`, `noiseSuppression: true`, and `autoGainControl: true` inside `navigator.mediaDevices.getUserMedia()`.
- On custom native apps, pair hardware AEC or WebRTC APM—never open raw uncancelled mic streams alongside loudspeakers.

2 Match Codec & Sample Rate End-to-End

Mismatched sample rates (sending 16kHz PCM into a 24kHz slot, or mixing `mu-law` and `a-law`) produce chipmunk pitch shifts or static hiss.

- **WebRTC:** Opus 48kHz negotiated automatically via SDP.
- **WebSocket Server:** `pcm16` at `24000 Hz` mono little-endian.
- **PSTN / SIP:** `g711_ulaw` (US) or `g711_alaw` (EU) at `8000 Hz`.

Deterministic Spoken Guardrails & Irreversible Action Gates

Guardrail Layer	What Goes Wrong Without It	Production Implementation Pattern
1. Two-Step Verbal Confirmation	Background TV speech or misheard digits trigger an unwanted refund, cancellation, or wire transfer.	Require explicit read-back inside tool logic: return a `needsConfirmation` state until the user confirms exact amount & target aloud.
2. Input & Topic Boundaries	Callers jailbreak a public voice kiosk into giving medical, legal, or off-brand advice.	Combine concise system boundaries with server-side classifier checks on transcripts before executing privileged MCP calls.
3. Quiet-Environment VAD Tuning	Coffeeshop clatter or street traffic keeps resetting `speech_started`, blocking the assistant from speaking.	Enable OpenAI’s `near_field` / `far_field` input noise reduction and raise VAD `threshold` from `0.50` to `0.65–0.72`.
4. Silent Warm Handoff	When two tool retries fail or caller frustration spikes, the bot loops apologetically forever.	Expose an `escalate_to_human` tool that transfers the SIP call (or opens live chat) while passing a structured summary card.

Pro Audio Tip: Never Mix Push-to-Talk Prompts with Server VAD

If your UI offers a physical “Hold to Talk” button, disable `turn_detection` (`null`) and commit the buffer explicitly on release. Leaving `server_vad` active under push-to-talk causes premature cutoffs whenever a user pauses mid-thought while still holding the button.

5 Real-World Copy-Paste Recipes

Use these five field-tested patterns to go from blank repository to production-grade voice and vision experiences—pairing the right model, transport, and tool mechanics for each job.

Recipe & Stack	Key APIs & Config	Copy-Paste Implementation Blueprint

1. Browser Support Copilot
Ephemeral WebRTC
	`POST /v1/realtime/client_secrets`
	`@openai/agents/realtime`
	`gpt-realtime-2.1`
		Add a `/api/realtime-token` Next.js route that mints a short-lived `ek_...` key using `OPENAI_API_KEY`. On button click in React, create `new RealtimeSession(agent)`, call `session.connect({ apiKey })`, and push DOM route state via `session.transport.sendEvent({ type: 'session.update', ... })`.

2. Live Camera Field Technician
Visual Diagnostics
	`addImage()`
	`getUserMedia(video)`
	`960px JPEG @ 0.5fps`
		Stream rear smartphone camera to a hidden `960x540` canvas every 2 seconds and push `session.addImage(dataUrl, { triggerResponse: false })`. Prompt the assistant to confirm wire colors and breaker labels visible in the latest frame before guiding the tech through one physical step at a time.

3. Twilio / Telnyx SIP Receptionist PSTN Telephony	`gpt-realtime-2.1-mini` `g711_ulaw` (8kHz) SIP / Media Stream	Point your Twilio or Telnyx inbound number to your call controller, negotiate `g711_ulaw` audio with zero resampling, and load a compact greeting prompt on `gpt-realtime-2.1-mini`. Expose `book_appointment` and `transfer_sip_extension` tools for instant call routing under $0.05/min.

4. Non-Blocking Order Lookup
Async Tool Flow
	Async `tool()`
	`sendFunctionCallOutput()`
	`sendMcpResponse()`
		Instruct the model: *"Before calling order tools, say Checking your order now."* Execute slow Postgres or Shopify lookups off the audio thread via async `tool()` (or push manually with `sendFunctionCallOutput(toolCall, json)`, and approve gated MCP calls with `sendMcpResponse(req, true)`).

5. Barge-In-Safe Language Tutor
Coaching & Pacing
	`server_vad`
	`silence_duration_ms: 850`
	`interrupt()`
		Raise `silence_duration_ms` from `500ms` to `850ms` so language learners have extra breathing room to finish thinking without getting cut off. Wire `audio_interrupted` to flush local WebAudio queues, and call `session.interrupt()` on any manual Stop button.

Combining Recipes 1 + 2 + 4: The Multimodal Support Super-App

Pair a browser **WebRTC** connection with optional screen-share `addImage()` snapshots and async `tool()` / `sendFunctionCallOutput()` account lookups. Your support agent can literally look at a customer's broken checkout screen, check their live Stripe invoice in the background, and walk them to a fix in under 45 seconds.

Latency Budgeting, Telemetry & Recovery

Sub-second voice feels effortless only when every hop in the audio path stays inside budget. Instrument these four timing checkpoints and build clean reconnect logic before exposing mobile users to spotty cellular handoffs.

End-to-End 800ms Perceived Turn-Around Budget

1. Capture + Network (~55ms)

Browser Opus frame encoding (~20ms) plus direct UDP WebRTC hop to OpenAI edge (~35ms). TCP proxies or distant TURN relays add 80–250ms.

2. VAD Endpoint (~350–500ms)

Controlled by `silence_duration_ms`. Dropping from `700ms` to `400ms` makes quick Q&A feel electric, while `850ms` suits reflective coaching.

3. First Audio Byte (~220ms)

Measured from `speech_stopped` to first received audio packet. Keep system prompts lean and push large docs behind async MCP tools.

📡 WebRTC ICE / TURN Firewall Diagnostics

Corporate VPNs and strict hospital/school Wi-Fi networks frequently block outbound UDP ports, silently breaking plain STUN peer connections:

- Log `peerConnection.iceConnectionState` and selected candidate pair (`host`, `srflx`, or `relay`).
- Configure a **TURN over TLS (port 443)** fallback ICE server so enterprise users fall back gracefully instead of hanging on `connecting`.

! Reconnect & Transcript Audit Trails

Mobile callers switch from Wi-Fi to 5G mid-call. Recover smoothly without forcing the user to repeat their story:

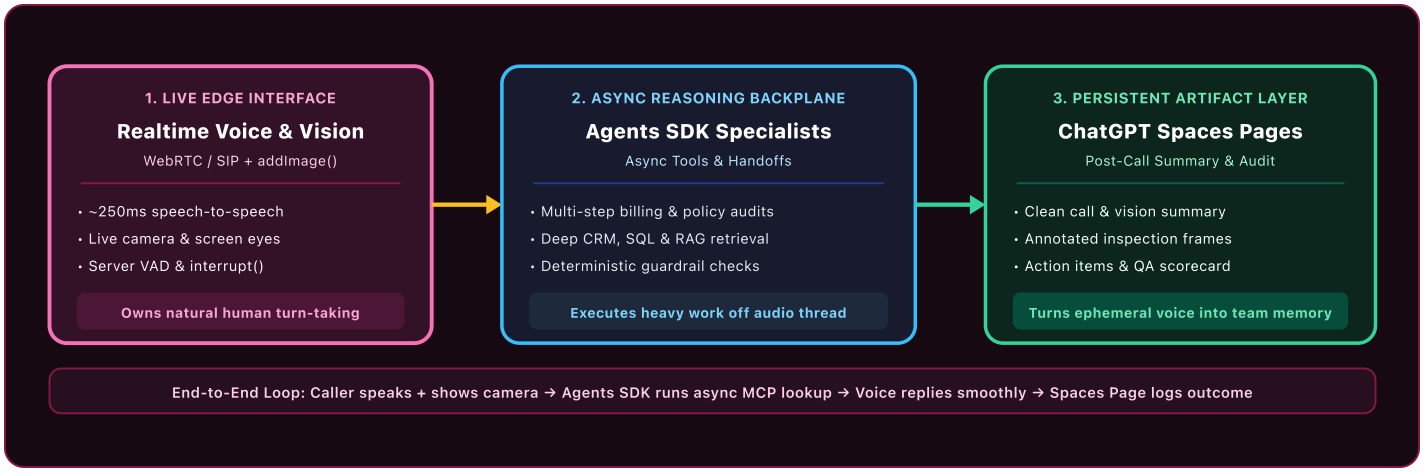
- Persist compacted user/assistant transcript turns and active tool results in local/server state as events arrive.
- On WebRTC disconnect, mint a fresh ephemeral token, reconnect, and replay the last 4–6 turns via conversation item injection.
- Scrub card numbers and SSNs from stored logs using audio/transcript PII redaction before archiving.

Track Time-to-First-Audio-Byte (TTFAB) Separately from Total Turn Duration

A 4-second spoken answer feels instant if audio starts in **240ms**, whereas a 1-second answer feels broken if silence stretches past **1.4 seconds** before the first syllable plays. Alert on P95 TTFAB spikes caused by bloated system instructions or oversized `addImage()` payloads.

Cross-Tool Synergy: Voice to Specialist Agents

Realtime models shine at low-latency conversation and live visual triage—not deep 30-second code or SQL reasoning. Pair your Realtime frontend with background **Agents SDK** specialists and post-call **ChatGPT Spaces** Pages.



How the Three Layers Cooperate on a Live Customer Inspection

1 Fast Voice Frontline + Background Specialist

A homeowner calls in over **WebRTC** and aims their phone camera at a blinking router via `addImage()`. `gpt-realtime-2.1` keeps them engaged smoothly while handing off serial-number warranty checks and firmware history to an **Agents SDK** specialist via async `tool()` / `sendFunctionCallOutput()`.

2 Zero-Effort Post-Call Documentation

The second the call ends, a webhook compiles the redacted transcript, verified hardware snapshot, and replacement ticket into a structured **ChatGPT Spaces Page**—so dispatch engineers or human supervisors inherit full context without replaying a 12-minute recording.

Architectural Golden Rule: Keep Heavy Reasoning Out of the Hot Audio Loop

Never force `gpt-realtime-2.1` to read 40 pages of policy PDFs inline during a live voice turn. Let a fast background Agents SDK worker query your vector store or database and return a two-sentence spoken summary back through async `tool()` or `sendFunctionCallOutput()`.

Troubleshooting Clinic & Pre-Flight Checklist

Almost every Realtime production hiccup maps to one of these seven transport, audio, or token issues. Run through this clinic and the six-point checklist before shipping to users.

Symptom / Error	Root Cause	10-Second Fix
1. WebRTC hangs at `connecting` behind corporate NAT	Strict firewall blocks outbound UDP STUN traffic, preventing peer-to-peer ICE negotiation.	Inspect `iceConnectionState` and add a **TURN over TLS (port 443)** relay fallback in your RTC configuration so corporate browsers connect over HTTPS/TCP.
2. Self-echo triggers false barge-in loops	Speaker playback bleeds back into the microphone and tricks Server VAD into firing `speech_started`.	Enable `echoCancellation: true` and `noiseSuppression: true` in `getUserMedia()`, test with headphones first, and raise VAD `threshold` to `0.60+`.
3. Exposed static API key flagged in client JS	Browser bundle passes permanent `sk-...` `OPENAI_API_KEY` directly to `session.connect()`.	Move `OPENAI_API_KEY` strictly to your backend and mint short-lived `ek_...` tokens via `POST /v1/realtime/client_secrets` per session.
4. SIP / G.711 audio sounds distorted or sped up	Sample rate or codec mismatch (e.g., feeding 8kHz `g711_ulaw` carrier audio into a 24kHz PCM16 session).	Explicitly declare `g711_ulaw` (or `g711_alaw`) on both input and output audio formats in session config, matching your carrier trunk verbatim.
5. `addImage()` payloads throttled or lagging audio	Client uploads uncompressed 4K PNGs at 15–30 fps over the WebRTC data channel.	Downscale canvas snapshots to `960px` JPEG at `0.72` quality, throttle to `0.5–1 fps`, and keep `triggerResponse: false` until needed.
6. Awkward dead air during tool calls	Tool handler awaits a 3-second database query synchronously before the model can speak again.	Prompt the model to speak a brief acknowledgment first, run slow queries inside async `tool()` or return manually via `sendFunctionCallOutput(toolCall, json)` (`sendMcpResponse()` is for MCP approvals).
7. Ephemeral token expires before user clicks Start	App pre-fetches `ek_...` on page load, but the user waits past its `expires_after` TTL (60s–600s).	Mint the ephemeral secret **on demand inside the Start click handler** right before calling `session.connect()`, never at initial page render.

6-Point Pre-Flight Checklist Before Shipping Realtime Voice & Vision

- ☑ **Zero Static Keys in Clients:** Confirmed only short-lived `ek_...` tokens from `/v1/realtime/client_secrets` reach browsers.
- ☑ **AEC & Noise Reduction Active:** Verified hardware echo cancellation works on laptop speakers with zero self-interruption loops.
- ☑ **Clean `interrupt()` Truncation:** Tested mid-sentence barge-in and confirmed local playback buffers flush immediately.
- ☑ **`addImage()` Throttled & Scaled:** Capped frames at ≤1 fps and ≤1024px JPEG with `triggerResponse: false`.
- ☑ **Async Tools Non-Blocking:** Verified slow tools speak an acknowledgment and resolve via async `tool()` / `sendFunctionCallOutput()`.
- ☑ **Firewall & Fallback Ready:** Validated TURN/TLS 443 traversal and warm human escalation with PII-redacted transcripts.