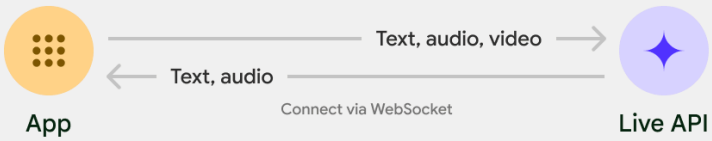
Get started with Live API

The Live API enables low-latency, real-time voice and video interactions with Gemini. It processes continuous streams of audio, video, or text to deliver immediate, human-like spoken responses, creating a natural conversational experience for your users.



Live API offers a comprehensive set of features such as Voice Activity Detection (/gemini-api/docs/live-guide#interruptions), tool use and function calling (/gemini-api/docs/live-tools), session management (/gemini-api/docs/live-session) (for managing long running conversations) and ephemeral tokens (/gemini-api/docs/ephemeral-tokens) (for secure client-sided authentication).

This page gets you up and running with examples and basic code samples.

Try the Live API in Google AI Studio (https://aistudio.google.com/live)

Choose an implementation approach

When integrating with Live API, you'll need to choose one of the following implementation approaches:

- **Server-to-server**: Your backend connects to the Live API using WebSockets (https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API). Typically, your c sends stream data (audio, video, text) to your server, which then forwards it to the Live API.
- **Client-to-server**: Your frontend code connects directly to the Live API using WebSockets (https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API) to stream data, bypassing your backend.

Note: Client-to-server generally offers better performance for streaming audio and video, since it bypasses the need to send the stream to your backend first. It's also easier to set up since you don't need to implement a proxy that sends data from your client to your server and then your server to the API. However, for production environments, in order to mitigate security risks, we recommend using ephemeral tokens (/gemini-api/docs/ephemeral-tokens) instead of standard API keys.

Partner integrations

To streamline the development of real-time audio and video apps, you can use a third-party integration that supports the Gemini Live API over WebRTC or WebSockets.

Pipecat by Daily

Create a real-time AI chatbot using Gemini Live and Pipecat.

(https://docs.pipecat.ai/guides/features/gemini-live)

LiveKit

Use the Gemini Live API with LiveKit Agents.

(https://docs.livekit.io/agents/models/realtime/plugins/gemini/)

Fishjam by Software Mansion

Create live video and audio streaming applications with Fishjam.

(https://docs.fishjam.io/tutorials/gemini-live-integration)

Agent Development Kit (ADK)

Implement the Live API with Agent Development Kit (ADK).

(https://google.github.io/adk-docs/streaming/)

Vision Agents by Stream

Build real-time voice and video AI applications with Vision Agents.

(https://visionagents.ai/integrations/gemini)

Voximplant

Connect inbound and outbound calls to Live API with Voximplant.

(https://voximplant.com/products/gemini-client)

Get started

✓ Microphone stream Audio file stream

This server-side example **streams audio from the microphone** and plays the returned audio. For complete end-to-end examples including a client application, see Example applications (#example-applications).

The input audio format should be in 16-bit PCM, 16kHz, mono format, and the received audio uses a sample rate of 24kHz.

Python (#python) JavaScript (#javascript)

Install helpers for audio streaming. Additional system-level dependencies might be required (`sox` for Mac/Windows or `ALSA` for Linux). Refer to the speaker (https://www.npmjs.com/package/speaker) and mic (https://www.npmjs.com/package/mic) docs for detailed installation steps.

```
$ npm install mic speaker
```

Note: Use headphones. This script uses the system default audio input and output, which often won't include echo cancellation. To prevent the model from interrupting itself, use headphones.

```
import { GoogleGenAI, Modality } from '@google/genai';
import mic from 'mic';
import Speaker from 'speaker';

const ai = new GoogleGenAI({});
// WARNING: Do not use API keys in client-side (browser based) applicati
// Consider using Ephemeral Tokens instead
// More information at: https://ai.google.dev/gemini-api/docs/ephemeral-

// --- Live API config ---
const model = 'gemini-2.5-flash-native-audio-preview-12-2025';
const config = {
  responseModalities: [Modality.AUDIO],
  systemInstruction: "You are a helpful and friendly AI assistant.",
};

async function live() {
  const responseQueue = [];
  const audioQueue = [];
  let speaker;

  async function waitMessage() {
    while (responseQueue.length === 0) {
```

```
    await new Promise((resolve) => setImmediate(resolve));
  }
  return responseQueue.shift();
}

function createSpeaker() {
  if (speaker) {
    process.stdin.unpipe(speaker);
    speaker.end();
  }
  speaker = new Speaker({
    channels: 1,
    bitDepth: 16,
    sampleRate: 24000,
  });
  speaker.on('error', (err) => console.error('Speaker error:', err));
  process.stdin.pipe(speaker);
}

async function messageLoop() {
  // Puts incoming messages in the audio queue.
  while (true) {
    const message = await waitMessage();
    if (message.serverContent && message.serverContent.interrupted) {
      // Empty the queue on interruption to stop playback
      audioQueue.length = 0;
      continue;
    }
    if (message.serverContent && message.serverContent.modelTurn && me
      for (const part of message.serverContent.modelTurn.parts) {
        if (part.inlineData && part.inlineData.data) {
          audioQueue.push(Buffer.from(part.inlineData.data, 'base64'))
        }
      }
    }
  }
}

async function playbackLoop() {
  // Plays audio from the audio queue.
  while (true) {
    if (audioQueue.length === 0) {
      if (speaker) {
        // Destroy speaker if no more audio to avoid warnings from spe
        process.stdin.unpipe(speaker);
        speaker.end();
        speaker = null;
      }
```

```
      await new Promise((resolve) => setImmediate(resolve));
    } else {
      if (!speaker) createSpeaker();
      const chunk = audioQueue.shift();
      await new Promise((resolve) => {
        speaker.write(chunk, () => resolve());
      });
    }
  }
}

// Start loops
messageLoop();
playbackLoop();

// Connect to Gemini Live API
const session = await ai.live.connect({
  model: model,
  config: config,
  callbacks: {
    onopen: () => console.log('Connected to Gemini Live API'),
    onmessage: (message) => responseQueue.push(message),
    onerror: (e) => console.error('Error:', e.message),
    onclose: (e) => console.log('Closed:', e.reason),
  },
});

// Setup Microphone for input
const micInstance = mic({
  rate: '16000',
  bitwidth: '16',
  channels: '1',
});
const micInputStream = micInstance.getAudioStream();

micInputStream.on('data', (data) => {
  // API expects base64 encoded PCM data
  session.sendRealtimeInput({
    audio: {
      data: data.toString('base64'),
      mimeType: "audio/pcm;rate=16000"
    }
  });
});

micInputStream.on('error', (err) => {
  console.error('Microphone error:', err);
});
```

```
  micInstance.start();
  console.log('Microphone started. Speak now...');
}

live().catch(console.error);
```

Example applications

Check out the following example applications that illustrate how to use Live API for end-to-end use cases:

- Live audio starter app (https://aistudio.google.com/apps/bundled/live_audio?showPreview=true&showCode=true&showAssistant=false) on AI Studio, using JavaScript libraries to connect to Live API and stream bidirectional audio through your microphone and speakers.
- See the Partner integrations (#partner-integrations) for additional examples and getting started guides.

What's next

- Read the full Live API Capabilities (/gemini-api/docs/live-guide) guide for key capabilities and configurations; including Voice Activity Detection and native audio features.
- Read the Tool use (/gemini-api/docs/live-tools) guide to learn how to integrate Live API with tools and function calling.
- Read the Session management (/gemini-api/docs/live-session) guide for managing long running conversations.
- Read the Ephemeral tokens (/gemini-api/docs/ephemeral-tokens) guide for secure authentication in client-to-server (#implementation-approach) applications.
- For more information about the underlying WebSockets API, see the WebSockets API reference (/api/live).

Last updated 2025-12-22 UTC.