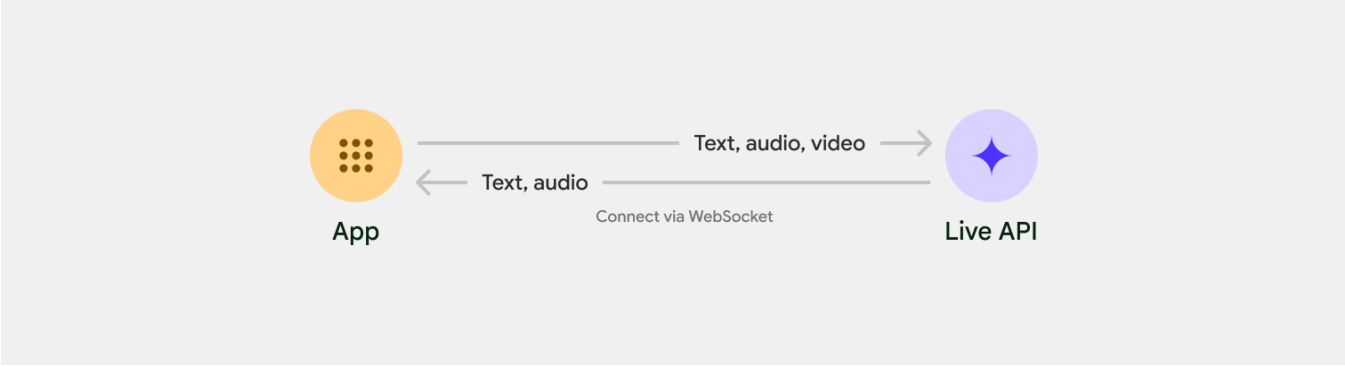
Veo 3 is now available in the Gemini API!

Learn more (https://developers.googleblog.com/en/veo-3-now-available-gemini-api/)

Get started with Live API

Preview: The Live API is in preview.

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.

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.
- Live API Python cookbook (https://colab.research.google.com/github/google-gemini/cookbook/blob/main/quickstarts/Get_started_LiveAPI.py) using Pyaudio that connects to Live API.

Partner integrations

If you prefer a simpler development process, you can use Daily (https://www.daily.co/products/gemini/multimodal-live-api/) or LiveKit (https://docs.livekit.io/agents/integrations/google/#multimodal-live-api). These are third-party partner platforms that have already integrated the Gemini Live API over the WebRTC protocol to streamline the development of real-time audio and video applications.

Before you begin building

There are two important decisions to make before you begin building with the Live API: choosing a model and choosing an implementation approach.

Choose an audio generation architecture

If you're building an audio-based use case, your choice of model determines the audio generation architecture used to create the audio response:

- **Native audio** (/gemini-api/docs/live-guide#native-audio-output)**:** This option provides the most natural and realistic-sounding speech and better multilingual performance. It also enables advanced features like affective (emotion-aware) dialogue (/gemini-api/docs/live-guide#affective-dialog), proactive audio (/gemini-api/docs/live-guide#proactive-audio) (where the model can decide to ignore or respond to certain inputs), and "thinking"

(/gemini-api/docs/live-guide#native-audio-output-thinking). Native audio is supported by the following native audio models (/gemini-api/docs/models#gemini-2.5-flash-native-audio):

  - `gemini-2.5-flash-preview-native-audio-dialog`
  - `gemini-2.5-flash-exp-native-audio-thinking-dialog`
- **Half-cascade audio**: This option uses a cascaded model architecture (native audio input and text-to-speech output). It offers better performance and reliability in production environments, especially with tool use (/gemini-api/docs/live-tools). Half-cascaded audio is supported by the following models:
  - `gemini-live-2.5-flash-preview`
  - `gemini-2.0-flash-live-001`

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 client 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.

Get started

This example ***reads a WAV file***, sends it in the correct format, and saves the received data as WAV file.

You can send audio by converting it to 16-bit PCM, 16kHz, mono format, and you can receive audio by setting `AUDIO` as response modality. The output uses a sample rate of 24kHz.

Python JavaScript (#javascript)
(#python)

```
# Test file: https://storage.googleapis.com/generativeai-downloads/data/160
# Install helpers for converting files: pip install librosa soundfile
import asyncio
import io
from pathlib import Path
import wave
from google import genai
from google.genai import types
import soundfile as sf
import librosa

client = genai.Client()

# Half cascade model:
# model = "gemini-live-2.5-flash-preview"

# Native audio output model:
model = "gemini-2.5-flash-preview-native-audio-dialog"

config = {
  "response_modalities": ["AUDIO"],
  "system_instruction": "You are a helpful assistant and answer in a friend
}

async def main():
    async with client.aio.live.connect(model=model, config=config) as sessi

        buffer = io.BytesIO()
        y, sr = librosa.load("sample.wav", sr=16000)
        sf.write(buffer, y, sr, format='RAW', subtype='PCM_16')
        buffer.seek(0)
        audio_bytes = buffer.read()

        # If already in correct format, you can use this:
        # audio_bytes = Path("sample.pcm").read_bytes()
```

```
        await session.send_realtime_input(
            audio=types.Blob(data=audio_bytes, mime_type="audio/pcm;rate=16
        )

        wf = wave.open("audio.wav", "wb")
        wf.setnchannels(1)
        wf.setsampwidth(2)
        wf.setframerate(24000)  # Output is 24kHz

        async for response in session.receive():
            if response.data is not None:
                wf.writeframes(response.data)

            # Un-comment this code to print audio data info
            # if response.server_content.model_turn is not None:
            #      print(response.server_content.model_turn.parts[0].inline

        wf.close()

if __name__ == "__main__":
    asyncio.run(main())
```

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 2025-07-08 UTC.