Live API capabilities guide

Preview: The Live API is in preview.

This is a comprehensive guide that covers capabilities and configurations available with the Live API. See Get started with Live API (/gemini-api/docs/live) page for a overview and sample code for common use cases.

Before you begin

- **Familiarize yourself with core concepts:** If you haven't already done so, read the Get started with Live API (/gemini-api/docs/live) page first. This will introduce you to the fundamental principles of the Live API, how it works, and the different implementation approaches (/gemini-api/docs/live#implementation-approach).
- **Try the Live API in AI Studio:** You may find it useful to try the Live API in Google AI Studio (https://aistudio.google.com/app/live) before you start building. To use the Live API in Google AI Studio, select **Stream**.

Establishing a connection

The following example shows how to create a connection with an API key:

Python (#python) JavaScript (#javascript)

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

const ai = new GoogleGenAI({});
const model = 'gemini-2.5-flash-native-audio-preview-12-2025';
const config = { responseModalities: [Modality.AUDIO] };

async function main() {

  const session = await ai.live.connect({
    model: model,
    callbacks: {
      onopen: function () {
        console.debug('Opened');
      },
      onmessage: function (message) {
        console.debug(message);
      },
      onerror: function (e) {
        console.debug('Error:', e.message);
      },
      onclose: function (e) {
        console.debug('Close:', e.reason);
      },
    },
    config: config,
  });

  console.debug("Session started");
  // Send content...

  session.close();
}

main();
```

Interaction modalities

The following sections provide examples and supporting context for the different input and output modalities available in Live API.

Sending and receiving audio

The most common audio example, **audio-to-audio**, is covered in the Getting started (/gemini-api/docs/live#audio-to-audio) guide.

Audio formats

Audio data in the Live API is always raw, little-endian, 16-bit PCM. Audio output always uses a sample rate of 24kHz. Input audio is natively 16kHz, but the Live API will resample if needed so any sample rate can be sent. To convey the sample rate of input audio, set the MIME type of each audio-containing Blob (/api/caching#Blob) to a value like **`audio/pcm;rate=16000`**.

Sending text

Here's how you can send text:

Python (#python)JavaScript (#javascript)

```
const message = 'Hello, how are you?';
session.sendClientContent({ turns: message, turnComplete: true });
```

Incremental content updates

Use incremental updates to send text input, establish session context, or restore session context. For short contexts you can send turn-by-turn interactions to represent the exact sequence of events:

Python (#python)JavaScript (#javascript)

```
let inputTurns = [
  { "role": "user", "parts": [{ "text": "What is the capital of France?"
  { "role": "model", "parts": [{ "text": "Paris" }] },
]

session.sendClientContent({ turns: inputTurns, turnComplete: false })

inputTurns = [{ "role": "user", "parts": [{ "text": "What is the capital

session.sendClientContent({ turns: inputTurns, turnComplete: true })
```

For longer contexts it's recommended to provide a single message summary to free up the context window for subsequent interactions. See Session Resumption (/gemini-api/docs/live-session#session-resumption) for another method for loading session context.

Audio transcriptions

In addition to the model response, you can also receive transcriptions of both the audio output and the audio input.

To enable transcription of the model's audio output, send `output_audio_transcription` in the setup config. The transcription language is inferred from the model's response.

Python (#python) JavaScript (#javascript)

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

const ai = new GoogleGenAI({});
const model = 'gemini-2.5-flash-native-audio-preview-12-2025';

const config = {
  responseModalities: [Modality.AUDIO],
  outputAudioTranscription: {}
};

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

  async function waitMessage() {
    let done = false;
    let message = undefined;
    while (!done) {
      message = responseQueue.shift();
      if (message) {
        done = true;
      } else {
        await new Promise((resolve) => setTimeout(resolve, 100));
      }
    }
    return message;
  }

  async function handleTurn() {
    const turns = [];
    let done = false;
    while (!done) {
      const message = await waitMessage();
      turns.push(message);
      if (message.serverContent && message.serverContent.turnComplete) {
        done = true;
      }
    }
    return turns;
  }

  const session = await ai.live.connect({
    model: model,
    callbacks: {
      onopen: function () {
        console.debug('Opened');
      },
      onmessage: function (message) {
        responseQueue.push(message);
      },
      onerror: function (e) {
        console.debug('Error:', e.message);
      },
      onclose: function (e) {
        console.debug('Close:', e.reason);
      },
    },
    config: config,
  });

  const inputTurns = 'Hello how are you?';
  session.sendClientContent({ turns: inputTurns });

  const turns = await handleTurn();

  for (const turn of turns) {
    if (turn.serverContent && turn.serverContent.outputTranscription) {
      console.debug('Received output transcription: %s\n', turn.serverCc
    }
  }

  session.close();
}

async function main() {
  await live().catch((e) => console.error('got error', e));
}

main();
```

To enable transcription of the model's audio input, send `input_audio_transcription` in setup config.

Python (#python) JavaScript (#javascript)

```
  import { GoogleGenAI, Modality } from '@google/genai';
  import * as fs from "node:fs";
```

```
import pkg from 'wavefile';
const { WaveFile } = pkg;

const ai = new GoogleGenAI({});
const model = 'gemini-2.5-flash-native-audio-preview-12-2025';

const config = {
  responseModalities: [Modality.AUDIO],
  inputAudioTranscription: {}
};

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

  async function waitMessage() {
    let done = false;
    let message = undefined;
    while (!done) {
      message = responseQueue.shift();
      if (message) {
        done = true;
      } else {
        await new Promise((resolve) => setTimeout(resolve, 100));
      }
    }
    return message;
  }

  async function handleTurn() {
    const turns = [];
    let done = false;
    while (!done) {
      const message = await waitMessage();
      turns.push(message);
      if (message.serverContent && message.serverContent.turnComplete) {
        done = true;
      }
    }
    return turns;
  }

  const session = await ai.live.connect({
    model: model,
    callbacks: {
      onopen: function () {
        console.debug('Opened');
      },
      onmessage: function (message) {
        responseQueue.push(message);
      },
      onerror: function (e) {
        console.debug('Error:', e.message);
      },
      onclose: function (e) {
        console.debug('Close:', e.reason);
      },
    },
    config: config,
  });

  // Send Audio Chunk
  const fileBuffer = fs.readFileSync("16000.wav");

  // Ensure audio conforms to API requirements (16-bit PCM, 16kHz, mono)
  const wav = new WaveFile();
  wav.fromBuffer(fileBuffer);
  wav.toSampleRate(16000);
  wav.toBitDepth("16");
  const base64Audio = wav.toBase64();

  // If already in correct format, you can use this:
  // const fileBuffer = fs.readFileSync("sample.pcm");
  // const base64Audio = Buffer.from(fileBuffer).toString('base64');

  session.sendRealtimeInput(
    {
      audio: {
        data: base64Audio,
        mimeType: "audio/pcm;rate=16000"
      }
    }
  );

  const turns = await handleTurn();
  for (const turn of turns) {
    if (turn.text) {
      console.debug('Received text: %s\n', turn.text);
    }
    else if (turn.data) {
      console.debug('Received inline data: %s\n', turn.data);
    }
    else if (turn.serverContent && turn.serverContent.inputTranscriptior
      console.debug('Received input transcription: %s\n', turn.serverCor
    }
  }
```

```
    session.close();
  }

  async function main() {
    await live().catch((e) => console.error('got error', e));
  }

  main();
```

Stream audio and video

To see an example of how to use the Live API in a streaming audio and video format, run the "Live API - Get Started" file in the cookbooks repository:

View on Colab
(https://github.com/google-gemini/cookbook/blob/main/quickstarts/Get_started_LiveAPI.py)

Change voice and language

Native audio output (#native-audio-output) models support any of the voices available for our Text-to-Speech (TTS) (/gemini-api/docs/speech-generation#voices) models. You can listen to all the voices in AI Studio (https://aistudio.google.com/app/live).

To specify a voice, set the voice name within the **`speechConfig`** object as part of the session configuration:

Python (#python)JavaScript
(#javascript)

```
  const config = {
    responseModalities: [Modality.AUDIO],
    speechConfig: { voiceConfig: { prebuiltVoiceConfig: { voiceName: "Kore
  };
```

Note: If you're using the **`generateContent`** API, the set of available voices is slightly different. See the audio generation guide (/gemini-api/docs/audio-generation#voices) for **`generateContent`** audio generation voices.

The Live API supports multiple languages (#supported-languages). Native audio output (#native-audio-output) models automatically choose the appropriate language and don't support explicitly setting the language code.

Native audio capabilities

Our latest models feature native audio output (/gemini-api/docs/models#gemini-2.5-flash-native-audio), which provides natural, realistic-sounding speech and improved multilingual performance. Native audio 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 intelligently decides when to respond to input), and "thinking" (/gemini-api/docs/live-guide#native-audio-output-thinking).

Affective dialog

This feature lets Gemini adapt its response style to the input expression and tone.

To use affective dialog, set the api version to `v1alpha` and set `enable_affective_dialog` to `true`in the setup message:

Python (#python) JavaScript (#javascript)

```
const ai = new GoogleGenAI({ httpOptions: {"apiVersion": "v1alpha"} });

const config = {
  responseModalities: [Modality.AUDIO],
  enableAffectiveDialog: true
};
```

Proactive audio

When this feature is enabled, Gemini can proactively decide not to respond if the content is not relevant.

To use it, set the api version to `v1alpha` and configure the `proactivity` field in the setup message and set `proactive_audio` to `true`:

Python (#python) JavaScript

```
const ai = new GoogleGenAI({ httpOptions: {"apiVersion": "v1alpha"} });

const config = {
  responseModalities: [Modality.AUDIO],
  proactivity: { proactiveAudio: true }
}
```

Thinking

The latest native audio output model **`gemini-2.5-flash-native-audio-preview-12-2025`** supports thinking capabilities (/gemini-api/docs/thinking), with dynamic thinking enabled by default.

The **`thinkingBudget`** parameter guides the model on the number of thinking tokens to use when generating a response. You can disable thinking by setting **`thinkingBudget`** to **`0`**. For more info on the **`thinkingBudget`** configuration details of the model, see the thinking budgets documentation (/gemini-api/docs/thinking#set-budget).

Python (#python) JavaScript (#javascript)

```
const model = 'gemini-2.5-flash-native-audio-preview-12-2025';
const config = {
  responseModalities: [Modality.AUDIO],
  thinkingConfig: {
    thinkingBudget: 1024,
  },
};

async function main() {

  const session = await ai.live.connect({
    model: model,
    config: config,
    callbacks: ...,
  });

  // Send audio input and receive audio

  session.close();
}
```

```
main();
```

Additionally, you can enable thought summaries by setting `includeThoughts` to `true` in your configuration. See thought summaries (/gemini-api/docs/thinking#summaries) for more info:

Python (#python) JavaScript (#javascript)

```
const model = 'gemini-2.5-flash-native-audio-preview-12-2025';
const config = {
  responseModalities: [Modality.AUDIO],
  thinkingConfig: {
    thinkingBudget: 1024,
    includeThoughts: true,
  },
};
```

Voice Activity Detection (VAD)

Voice Activity Detection (VAD) allows the model to recognize when a person is speaking. This is essential for creating natural conversations, as it allows a user to interrupt the model at any time.

When VAD detects an interruption, the ongoing generation is canceled and discarded. Only the information already sent to the client is retained in the session history. The server then sends a `BidiGenerateContentServerContent` (/api/live#bidigeneratecontentservercontent) message to report the interruption.

The Gemini server then discards any pending function calls and sends a `BidiGenerateContentServerContent` message with the IDs of the canceled calls.

Python (#python) JavaScript (#javascript)

```
const turns = await handleTurn();

for (const turn of turns) {
  if (turn.serverContent && turn.serverContent.interrupted) {
    // The generation was interrupted
```

```
      // If realtime playback is implemented in your application,
      // you should stop playing audio and clear queued playback here.
    }
  }
```

Automatic VAD

By default, the model automatically performs VAD on a continuous audio input stream. VAD can be configured with the `realtimeInputConfig.automaticActivityDetection` (/api/live#RealtimeInputConfig.AutomaticActivityDetection) field of the setup configuration (/api/live#BidiGenerateContentSetup).

When the audio stream is paused for more than a second (for example, because the user switched off the microphone), an `audioStreamEnd` (/api/live#BidiGenerateContentRealtimeInput.FIELDS.bool.BidiGenerateContentRealtimeInput.audio_stream_end) event should be sent to flush any cached audio. The client can resume sending audio data at any time.

Python (#python) JavaScript (#javascript)

```
// example audio file to try:
// URL = "https://storage.googleapis.com/generativeai-downloads/data/hel
// !wget -q $URL -O sample.pcm
import { GoogleGenAI, Modality } from '@google/genai';
import * as fs from "node:fs";

const ai = new GoogleGenAI({});
const model = 'gemini-live-2.5-flash-preview';
const config = { responseModalities: [Modality.TEXT] };

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

  async function waitMessage() {
    let done = false;
    let message = undefined;
    while (!done) {
      message = responseQueue.shift();
      if (message) {
        done = true;
      } else {
        await new Promise((resolve) => setTimeout(resolve, 100));
      }
    }
    return message;
  }

  async function handleTurn() {
    const turns = [];
    let done = false;
    while (!done) {
      const message = await waitMessage();
      turns.push(message);
      if (message.serverContent && message.serverContent.turnComplete) {
        done = true;
      }
    }
    return turns;
  }

  const session = await ai.live.connect({
    model: model,
    callbacks: {
      onopen: function () {
        console.debug('Opened');
      },
      onmessage: function (message) {
        responseQueue.push(message);
      },
      onerror: function (e) {
        console.debug('Error:', e.message);
      },
      onclose: function (e) {
        console.debug('Close:', e.reason);
      },
    },
    config: config,
  });

  // Send Audio Chunk
  const fileBuffer = fs.readFileSync("sample.pcm");
  const base64Audio = Buffer.from(fileBuffer).toString('base64');

  session.sendRealtimeInput(
    {
      audio: {
        data: base64Audio,
        mimeType: "audio/pcm;rate=16000"
      }
    }

  );

  // if stream gets paused, send:
  // session.sendRealtimeInput({ audioStreamEnd: true })

  const turns = await handleTurn();
  for (const turn of turns) {
    if (turn.text) {
      console.debug('Received text: %s\n', turn.text);
    }
    else if (turn.data) {
      console.debug('Received inline data: %s\n', turn.data);
    }
  }

  session.close();
}

async function main() {
  await live().catch((e) => console.error('got error', e));
}

main();
```

With **`send_realtime_input`**, the API will respond to audio automatically based on VAD. While **`send_client_content`** adds messages to the model context in order, **`send_realtime_input`** is optimized for responsiveness at the expense of deterministic ordering.

Automatic VAD configuration

For more control over the VAD activity, you can configure the following parameters. See API reference (/api/live#automaticactivitydetection) for more info.

Python (#python) JavaScript (#javascript)

```
import { GoogleGenAI, Modality, StartSensitivity, EndSensitivity } from

const config = {
  responseModalities: [Modality.TEXT],
  realtimeInputConfig: {
    automaticActivityDetection: {
      disabled: false, // default
      startOfSpeechSensitivity: StartSensitivity.START_SENSITIVITY_LOW,
      endOfSpeechSensitivity: EndSensitivity.END_SENSITIVITY_LOW,
      prefixPaddingMs: 20,
      silenceDurationMs: 100,
    }
  }
};
```

Disable automatic VAD

Alternatively, the automatic VAD can be disabled by setting **`realtimeInputConfig.automaticActivityDetection.disabled`** to **`true`** in the setup message. In this configuration the client is responsible for detecting user speech and sending **`activityStart`** (/api/live#BidiGenerateContentRealtimeInput.FIELDS.BidiGenerateContentRealtimeInput.ActivityStart.BidiGenerateContentRealtimeInput.activity_start) and **`activityEnd`** (/api/live#BidiGenerateContentRealtimeInput.FIELDS.BidiGenerateContentRealtimeInput.ActivityEnd.BidiGenerateContentRealtimeInput.activity_end) messages at the appropriate times. An **`audioStreamEnd`** isn't sent in this configuration. Instead, any interruption of the stream is marked by an **`activityEnd`** message.

Python (#python) JavaScript (#javascript)

```
  const config = {
    responseModalities: [Modality.TEXT],
    realtimeInputConfig: {
      automaticActivityDetection: {
        disabled: true,
      }
    }
  };

  session.sendRealtimeInput({ activityStart: {} })

  session.sendRealtimeInput(
    {
      audio: {
        data: base64Audio,
        mimeType: "audio/pcm;rate=16000"
      }
    }

  );

  session.sendRealtimeInput({ activityEnd: {} })
```

Token count

You can find the total number of consumed tokens in the usageMetadata (/api/live#usagemetadata) field of the returned server message.

Python (#python)JavaScript (#javascript)

```
const turns = await handleTurn();

for (const turn of turns) {
  if (turn.usageMetadata) {
    console.debug('Used %s tokens in total. Response token breakdown:\n'

    for (const detail of turn.usageMetadata.responseTokensDetails) {
      console.debug('%s\n', detail);
    }
  }
}
```

Media resolution

You can specify the media resolution for the input media by setting the **`mediaResolution`** field as part of the session configuration:

Python (#python)JavaScript (#javascript)

```
import { GoogleGenAI, Modality, MediaResolution } from '@google/genai';

const config = {
    responseModalities: [Modality.TEXT],
    mediaResolution: MediaResolution.MEDIA_RESOLUTION_LOW,
  };
```

Limitations

Consider the following limitations of the Live API when you plan your project.

Response modalities

You can only set one response modality (`TEXT` or `AUDIO`) per session in the session configuration. Setting both results in a config error message. This means that you can configure the model to respond with either text or audio, but not both in the same session.

Client authentication

The Live API only provides server-to-server authentication by default. If you're implementing your Live API application using a client-to-server approach (/gemini-api/docs/live#implementation-approach), you need to use ephemeral tokens (/gemini-api/docs/ephemeral-tokens) to mitigate security risks.

Session duration

Audio-only sessions are limited to 15 minutes, and audio plus video sessions are limited to 2 minutes. However, you can configure different session management techniques (/gemini-api/docs/live-session) for unlimited extensions on session duration.

Context window

A session has a context window limit of:

- 128k tokens for native audio output (#native-audio-output) models
- 32k tokens for other Live API models

Supported languages

Live API supports the following 70 languages.

Note: Native audio output (#native-audio-output) models can switch between languages naturally during conversation. You can also restrict the languages it speaks in by specifying it in the system instructions.

Language	BCP-47 Code	Language	BCP-47 Code
Afrikaans	`af`	Kannada	`kn`
Albanian	`sq`	Kazakh	`kk`
Amharic	`am`	Khmer	`km`
Arabic	`ar`	Korean	`ko`
Armenian	`hy`	Lao	`lo`
Assamese	`as`	Latvian	`lv`
Azerbaijani	`az`	Lithuanian	`lt`
Basque	`eu`	Macedonian	`mk`
Belarusian	`be`	Malay	`ms`
Bengali	`bn`	Malayalam	`ml`
Bosnian	`bs`	Marathi	`mr`
Bulgarian	`bg`	Mongolian	`mn`
Catalan	`ca`	Nepali	`ne`
Chinese	`zh`	Norwegian	`no`
Croatian	`hr`	Odia	`or`

Language	BCP-47 Code	Language	BCP-47 Code
Czech	`cs`	Polish	`pl`
Danish	`da`	Portuguese	`pt`
Dutch	`nl`	Punjabi	`pa`
English	`en`	Romanian	`ro`
Estonian	`et`	Russian	`ru`
Filipino	`fil`	Serbian	`sr`
Finnish	`fi`	Slovak	`sk`
French	`fr`	Slovenian	`sl`
Galician	`gl`	Spanish	`es`
Georgian	`ka`	Swahili	`sw`
German	`de`	Swedish	`sv`
Greek	`el`	Tamil	`ta`
Gujarati	`gu`	Telugu	`te`
Hebrew	`iw`	Thai	`th`
Hindi	`hi`	Turkish	`tr`
Hungarian	`hu`	Ukrainian	`uk`
Icelandic	`is`	Urdu	`ur`

Language	BCP-47 Code	Language	BCP-47 Code
Indonesian	`id`	Uzbek	`uz`
Italian	`it`	Vietnamese	`vi`
Japanese	`ja`	Zulu	`zu`

What's next

- Read the Tool Use (/gemini-api/docs/live-tools) and Session Management (/gemini-api/docs/live-session) guides for essential information on using the Live API effectively.
- Try the Live API in Google AI Studio (https://aistudio.google.com/app/live).
- For more info about the Live API models, see Gemini 2.5 Flash Native Audio (/gemini-api/docs/models#gemini-2.5-flash-native-audio) on the Models page.
- Try more examples in the Live API cookbook (https://colab.research.google.com/github/google-gemini/cookbook/blob/main/quickstarts/Get_started_LiveAPI.ipynb) , the Live API Tools cookbook (https://colab.research.google.com/github/google-gemini/cookbook/blob/main/quickstarts/Get_started_LiveAPI_tools.ipynb) , and the Live API Get Started script (https://github.com/google-gemini/cookbook/blob/main/quickstarts/Get_started_LiveAPI.py).