Veo 3 is now available in the Gemini API!

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

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 distinction between the different models (/gemini-api/docs/live#audio-generation) and their corresponding audio generation methods (native audio (#native-audio-output) or half-cascade).
- **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 JavaScript (#javascript)
(#python)

```
import asyncio
from google import genai
```

```
client = genai.Client()

model = "gemini-live-2.5-flash-preview"
config = {"response_modalities": ["TEXT"]}

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

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

Note: You can only set one modality (/gemini-api/docs/live#response-modalities) in the **`response_modalities`** field. This means that you can configure the model to respond with either text or audio, but not both in the same session.

Interaction modalities

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

Sending and receiving text

Here's how you can send and receive text:

Python JavaScript (#javascript)
(#python)

```
import asyncio
from google import genai

client = genai.Client()
model = "gemini-live-2.5-flash-preview"

config = {"response_modalities": ["TEXT"]}
```

```
async def main():
    async with client.aio.live.connect(model=model, config=config) as sessi
        message = "Hello, how are you?"
        await session.send_client_content(
            turns={"role": "user", "parts": [{"text": message}]}, turn_comp
        )

        async for response in session.receive():
            if response.text is not None:
                print(response.text, end="")

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

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:

PythonJavaScript (#javascript)
(#python)

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

await session.send_client_content(turns=turns, turn_complete=False)

turns = [{"role": "user", "parts": [{"text": "What is the capital of Germar

await session.send_client_content(turns=turns, turn_complete=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.

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.

Here's an **audio-to-text** example that reads a WAV file, sends it in the correct format and receives text output:

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
from google import genai
from google.genai import types
import soundfile as sf
import librosa

client = genai.Client()
model = "gemini-live-2.5-flash-preview"

config = {"response_modalities": ["TEXT"]}

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
        )

        async for response in session.receive():
            if response.text is not None:
                print(response.text)
```

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

And here is a **text-to-audio** example. You can receive audio by setting `AUDIO` as response modality. This example saves the received data as WAV file:

PythonJavaScript (#javascript)
(#python)

```
import asyncio
import wave
from google import genai

client = genai.Client()
model = "gemini-live-2.5-flash-preview"

config = {"response_modalities": ["AUDIO"]}

async def main():
    async with client.aio.live.connect(model=model, config=config) as sessi
        wf = wave.open("audio.wav", "wb")
        wf.setnchannels(1)
        wf.setsampwidth(2)
        wf.setframerate(24000)

        message = "Hello how are you?"
        await session.send_client_content(
            turns={"role": "user", "parts": [{"text": message}]}, turn_comp
        )

        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())
```

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`**.

Audio transcriptions

You can enable transcription of the model's audio output by sending **`output_audio_transcription`** in the setup config. The transcription language is inferred from the model's response.

Python JavaScript (#javascript)
(#python)

```
import asyncio
from google import genai
from google.genai import types

client = genai.Client()
model = "gemini-live-2.5-flash-preview"

config = {"response_modalities": ["AUDIO"],
        "output_audio_transcription": {}
}

async def main():
    async with client.aio.live.connect(model=model, config=config) as sessi
        message = "Hello? Gemini are you there?"

        await session.send_client_content(
            turns={"role": "user", "parts": [{"text": message}]}, turn_comp
        )

        async for response in session.receive():
            if response.server_content.model_turn:
                print("Model turn:", response.server_content.model_turn)
```

```
            if response.server_content.output_transcription:
                print("Transcript:", response.server_content.output_transcr

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

You can enable transcription of the audio input by sending `input_audio_transcription` in setup config.

Python JavaScript (#javascript)
(#python)

```
import asyncio
from pathlib import Path
from google import genai
from google.genai import types

client = genai.Client()
model = "gemini-live-2.5-flash-preview"

config = {
    "response_modalities": ["TEXT"],
    "input_audio_transcription": {},
}

async def main():
    async with client.aio.live.connect(model=model, config=config) as sessi
        audio_data = Path("16000.pcm").read_bytes()

        await session.send_realtime_input(
            audio=types.Blob(data=audio_data, mime_type='audio/pcm;rate=160
        )

        async for msg in session.receive():
            if msg.server_content.input_transcription:
                print('Transcript:', msg.server_content.input_transcription

if __name__ == "__main__":
    asyncio.run(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://colab.research.google.com/github/google-gemini/cookbook/blob/main/quickstarts/Get_started_LiveAPI.py)

Change voice and language

The Live API models each support a different set of voices. Half-cascade supports Puck, Charon, Kore, Fenrir, Aoede, Leda, Orus, and Zephyr. Native audio supports a much longer list (identical to the TTS model list (/gemini-api/docs/speech-generation#voices)). 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 JavaScript (#javascript)
(#python)

```
config = {
    "response_modalities": ["AUDIO"],
    "speech_config": {
        "voice_config": {"prebuilt_voice_config": {"voice_name": "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).

To change the language, set the language code within the **`speechConfig`** object as part of the session configuration:

PythonJavaScript (#javascript)
(#python)

```
config = {
    "response_modalities": ["AUDIO"],
    "speech_config": {
        "language_code": "de-DE"
    }
}
```

Note: Native audio output (#native-audio-output) models automatically choose the appropriate language and don't support explicitly setting the language code.

Native audio capabilities

The following capabilities are only available with native audio. You can learn more about native audio in Choose a model and audio generation (/gemini-api/docs/models).

Note: Native audio models currently have limited tool use support. See Overview of supported tools (/gemini-api/docs/live-tools#tools-overview) for details.

How to use native audio output

To use native audio output, configure one of the native audio models (/gemini-api/docs/models#gemini-2.5-flash-native-audio) and set `response_modalities` to `AUDIO`.

See Send and receive audio (/gemini-api/docs/live#audio-to-audio) for a full example.

PythonJavaScript (#javascript)
(#python)

```
model = "gemini-2.5-flash-preview-native-audio-dialog"
config = types.LiveConnectConfig(response_modalities=["AUDIO"])

async with client.aio.live.connect(model=model, config=config) as session:
```

```
    # Send audio input and receive audio
```

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 JavaScript (#javascript)
(#python)

```
client = genai.Client(http_options={"api_version": "v1alpha"})

config = types.LiveConnectConfig(
    response_modalities=["AUDIO"],
    enable_affective_dialog=True
)
```

Note that affective dialog is currently only supported by the native audio output models.

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 JavaScript (#javascript)
(#python)

```
client = genai.Client(http_options={"api_version": "v1alpha"})

config = types.LiveConnectConfig(
    response_modalities=["AUDIO"],
    proactivity={'proactive_audio': True}
```

```
)
```

Note that proactive audio is currently only supported by the native audio output models.

Native audio output with thinking

Native audio output supports thinking capabilities (/gemini-api/docs/thinking), available via a separate model `gemini-2.5-flash-exp-native-audio-thinking-dialog`.

See Send and receive audio (/gemini-api/docs/live#audio-to-audio) for a full example.

Python JavaScript (#javascript)
(#python)

```
model = "gemini-2.5-flash-exp-native-audio-thinking-dialog"
config = types.LiveConnectConfig(response_modalities=["AUDIO"])

async with client.aio.live.connect(model=model, config=config) as session:
    # Send audio input and receive audio
```

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 JavaScript (#javascript)
(#python)

```
async for response in session.receive():
    if response.server_content.interrupted is True:
        # 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 JavaScript (#javascript) (#python)

```
# example audio file to try:
# URL = "https://storage.googleapis.com/generativeai-downloads/data/hello_a
# !wget -q $URL -O sample.pcm
import asyncio
from pathlib import Path
from google import genai
from google.genai import types

client = genai.Client()
model = "gemini-live-2.5-flash-preview"

config = {"response_modalities": ["TEXT"]}

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

```
        audio_bytes = Path("sample.pcm").read_bytes()

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

        # if stream gets paused, send:
        # await session.send_realtime_input(audio_stream_end=True)

        async for response in session.receive():
            if response.text is not None:
                print(response.text)

if __name__ == "__main__":
    asyncio.run(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 JavaScript (#javascript)
(#python)

```
from google.genai import types

config = {
    "response_modalities": ["TEXT"],
    "realtime_input_config": {
        "automatic_activity_detection": {
            "disabled": False, # default
            "start_of_speech_sensitivity": types.StartSensitivity.START_SEN
            "end_of_speech_sensitivity": types.EndSensitivity.END_SENSITIVI
            "prefix_padding_ms": 20,
            "silence_duration_ms": 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 JavaScript (#javascript)
(#python)

```
config = {
    "response_modalities": ["TEXT"],
    "realtime_input_config": {"automatic_activity_detection": {"disabled":
}

async with client.aio.live.connect(model=model, config=config) as session:
    # ...
    await session.send_realtime_input(activity_start=types.ActivityStart())
    await session.send_realtime_input(
        audio=types.Blob(data=audio_bytes, mime_type="audio/pcm;rate=16000"
    )
    await session.send_realtime_input(activity_end=types.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 JavaScript (#javascript)
(#python)

```
async for message in session.receive():
    # The server will periodically send messages that include UsageMetadata
    if message.usage_metadata:
        usage = message.usage_metadata
        print(
            f"Used {usage.total_token_count} tokens in total. Response toke
        )
        for detail in usage.response_tokens_details:
            match detail:
                case types.ModalityTokenCount(modality=modality, token_coun
                    print(f"{modality}: {count}")
```

Media resolution

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

Python JavaScript (#javascript)
(#python)

```
from google.genai import types

config = {
    "response_modalities": ["AUDIO"],
    "media_resolution": types.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 languages.

Note: Native audio output (#native-audio-output) models automatically choose the appropriate language and don't support explicitly setting the language code.

Language	BCP-47 Code	Language	BCP-47 Code
German (Germany)	`de-DE`	English (Australia)*	`en-AU`
English (UK)*	`en-GB`	English (India)	`en-IN`
English (US)	`en-US`	Spanish (US)	`es-US`
French (France)	`fr-FR`	Hindi (India)	`hi-IN`
Portuguese (Brazil)	`pt-BR`	Arabic (Generic)	`ar-XA`
Spanish (Spain)*	`es-ES`	French (Canada)*	`fr-CA`
Indonesian (Indonesia)	`id-ID`	Italian (Italy)	`it-IT`
Japanese (Japan)	`ja-JP`	Turkish (Turkey)	`tr-TR`
Vietnamese (Vietnam)	`vi-VN`	Bengali (India)	`bn-IN`
Gujarati (India)*	`gu-IN`	Kannada (India)*	`kn-IN`
Marathi (India)	`mr-IN`	Malayalam (India)*	`ml-IN`
Tamil (India)	`ta-IN`	Telugu (India)	`te-IN`
Dutch (Netherlands)	`nl-NL`	Korean (South Korea)	`ko-KR`
Mandarin Chinese (China)*	`cmn-CN`	Polish (Poland)	`pl-PL`
Russian (Russia)	`ru-RU`	Thai (Thailand)	`th-TH`

Languages marked with an asterisk (*) *are not available for Native audio* (#native-audio-output).

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 2.0 Flash Live (/gemini-api/docs/models#live-api) and 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://colab.research.google.com/github/google-gemini/cookbook/blob/main/quickstarts/Get_started_LiveAPI.py).

Last updated 2025-07-16 UTC.