Tool use with Live API

Tool use allows Live API to go beyond just conversation by enabling it to perform actions in the real-world and pull in external context while maintaining a real time connection. You can define tools such as Function calling (/gemini-api/docs/function-calling) and Google Search (/gemini-api/docs/grounding) with the Live API.

Overview of supported tools

Here's a brief overview of the available tools for Live API models:

Tool	`gemini-2.5-flash-native-audio-preview-12-2025`
Search	Yes
Function calling	Yes
Google Maps	No
Code execution	No
URL context	No

Function calling

Live API supports function calling, just like regular content generation requests. Func calling lets the Live API interact with external data and programs, greatly increasing v your applications can accomplish.

You can define function declarations as part of the session configuration. After receiving tool calls, the client should respond with a list of `FunctionResponse` objects using the `session.send_tool_response` method.

See the Function calling tutorial (/gemini-api/docs/function-calling) to learn more.

Note: Unlike the `generateContent` API, the Live API doesn't support automatic tool response handling. You must handle tool responses manually in your client code.

Python (#python)JavaScript (#javascript)

```
import { GoogleGenAI, Modality } from '@google/genai';
import * as fs from "node:fs";
import pkg from 'wavefile';  // npm install wavefile
const { WaveFile } = pkg;

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

// Simple function definitions
const turn_on_the_lights = { name: "turn_on_the_lights" } // , descripti
const turn_off_the_lights = { name: "turn_off_the_lights" }

const tools = [{ functionDeclarations: [turn_on_the_lights, turn_off_the

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

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;
      } else if (message.toolCall) {
        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 = 'Turn on the lights please';
  session.sendClientContent({ turns: inputTurns });

  let turns = await handleTurn();

  for (const turn of turns) {
    if (turn.toolCall) {
      console.debug('A tool was called');
      const functionResponses = [];
      for (const fc of turn.toolCall.functionCalls) {
        functionResponses.push({
          id: fc.id,
          name: fc.name,
          response: { result: "ok" } // simple, hard-coded function resp
        });
      }

      console.debug('Sending tool response...\n');
```

```
        session.sendToolResponse({ functionResponses: functionResponses })
      }
    }

    // Check again for new messages
    turns = await handleTurn();

    // Combine audio data strings and save as wave file
    const combinedAudio = turns.reduce((acc, turn) => {
        if (turn.data) {
            const buffer = Buffer.from(turn.data, 'base64');
            const intArray = new Int16Array(buffer.buffer, buffer.byteOffs
            return acc.concat(Array.from(intArray));
        }
        return acc;
    }, []);

    const audioBuffer = new Int16Array(combinedAudio);

    const wf = new WaveFile();
    wf.fromScratch(1, 24000, '16', audioBuffer);  // output is 24kHz
    fs.writeFileSync('audio.wav', wf.toBuffer());

    session.close();
  }

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

  main();
```

From a single prompt, the model can generate multiple function calls and the code necessary to chain their outputs. This code executes in a sandbox environment, generating subsequent BidiGenerateContentToolCall (/api/live#bidigeneratecontenttoolcall) messages.

Asynchronous function calling

Function calling executes sequentially by default, meaning execution pauses until the results of each function call are available. This ensures sequential processing, which means you won't be able to continue interacting with the model while the functions are being run.

If you don't want to block the conversation, you can tell the model to run the functions asynchronously. To do so, you first need to add a `behavior` to the function definitions:

Python (#python) JavaScript (#javascript)

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

// Non-blocking function definitions
const turn_on_the_lights = {name: "turn_on_the_lights", behavior: Behavi

// Blocking function definitions
const turn_off_the_lights = {name: "turn_off_the_lights"}

const tools = [{ functionDeclarations: [turn_on_the_lights, turn_off_the
```

`NON-BLOCKING` ensures the function runs asynchronously while you can continue interacting with the model.

Then you need to tell the model how to behave when it receives the `FunctionResponse` using the `scheduling` parameter. It can either:

- Interrupt what it's doing and tell you about the response it got right away (`scheduling="INTERRUPT"`),
- Wait until it's finished with what it's currently doing (`scheduling="WHEN_IDLE"`),
- Or do nothing and use that knowledge later on in the discussion (`scheduling="SILENT"`)

Python (#python) JavaScript (#javascript)

```
import { GoogleGenAI, Modality, Behavior, FunctionResponseScheduling } f

// for a non-blocking function definition, apply scheduling in the funct
const functionResponse = {
  id: fc.id,
  name: fc.name,
  response: {
    result: "ok",
    scheduling: FunctionResponseScheduling.INTERRUPT  // Can also be WHE
  }
}
```

Grounding with Google Search

You can enable Grounding with Google Search as part of the session configuration. This increases the Live API's accuracy and prevents hallucinations. See the Grounding tutorial (/gemini-api/docs/grounding) to learn more.

Python (#python) JavaScript (#javascript)

```
import { GoogleGenAI, Modality } from '@google/genai';
import * as fs from "node:fs";
import pkg from 'wavefile';  // npm install wavefile
const { WaveFile } = pkg;

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

const tools = [{ googleSearch: {} }]
const config = {
  responseModalities: [Modality.AUDIO],
  tools: tools
}

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;
      } else if (message.toolCall) {
        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 = 'When did the last Brazil vs. Argentina soccer matc
  session.sendClientContent({ turns: inputTurns });

  let turns = await handleTurn();

  let combinedData = '';
  for (const turn of turns) {
    if (turn.serverContent && turn.serverContent.modelTurn && turn.serve
      for (const part of turn.serverContent.modelTurn.parts) {
        if (part.executableCode) {
          console.debug('executableCode: %s\n', part.executableCode.code
        }
        else if (part.codeExecutionResult) {
          console.debug('codeExecutionResult: %s\n', part.codeExecutionR
        }
        else if (part.inlineData && typeof part.inlineData.data === 'str
          combinedData += atob(part.inlineData.data);
        }
      }
    }
  }

  // Convert the base64-encoded string of bytes into a Buffer.
  const buffer = Buffer.from(combinedData, 'binary');

  // The buffer contains raw bytes. For 16-bit audio, we need to interpr
  const intArray = new Int16Array(buffer.buffer, buffer.byteOffset, buff

  const wf = new WaveFile();
  // The API returns 16-bit PCM audio at a 24kHz sample rate.
  wf.fromScratch(1, 24000, '16', intArray);
  fs.writeFileSync('audio.wav', wf.toBuffer());

  session.close();
}

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

main();
```

Combining multiple tools

You can combine multiple tools within the Live API, increasing your application's capabilities even more:

Python (#python)JavaScript (#javascript)

```
const prompt = `Hey, I need you to do two things for me.

1. Use Google Search to look up information about the largest earthquake
2. Then turn on the lights

Thanks!
`
```

```
const tools = [
  { googleSearch: {} },
  { functionDeclarations: [turn_on_the_lights, turn_off_the_lights] }
]

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

// ... remaining model call
```

What's next

- Check out more examples of using tools with the Live API in the Tool use cookbook (https://colab.research.google.com/github/google-gemini/cookbook/blob/main/quickstarts/Get_started_LiveAPI_tools.ipynb)
  .
- Get the full story on features and configurations from the Live API Capabilities guide (/gemini-api/docs/live-guide).

Last updated 2025-12-18 UTC.