Tutorial 15: Live API and Audio - Real-Time Voice Interactions

Difficulty: advanced

Reading Time: 2 hours

Tags: advanced, live-api, audio, voice, real-time

Description: Create voice-enabled agents using Gemini's Live API for real-time audio streaming and voice-to-voice conversations.

:::info UPDATED - ADK WEB FOCUSED APPROACH

This tutorial has been streamlined to focus on the working method for Live API: ADK Web Interface.

Key Updates (January 12, 2025):
- ✅ **Recommended Approach**: Use `adk web` for Live API bidirectional streaming
- ✅ **Why**: `runner.run_live()` requires WebSocket server context (works in `adk web`, not standalone scripts)
- ✅ **Core Components**: Agent definition and audio utilities for programmatic use
- ✅ **Simplified**: Removed non-working standalone demo scripts
- ✅ **Focus**: Single clear path - start ADK web server and use browser interface

Working implementation available: [Tutorial 15 Implementation](https://github.com/raphaelmansuy/adk_training/tree/main/tutorial_implementation/tutorial15) (https://github.com/raphaelmansuy/adk_training/tree/main/tutorial_implementation/tutorial15)

Quick Start:

```
cd tutorial_implementation/tutorial15
make setup  # Install dependencies
make dev    # Start ADK web interface
# Open http://localhost:8000 and select 'voice_assistant'
```

:::

Tutorial 15: Live API & Bidirectional Streaming with Audio

Goal: Master the Live API for bidirectional streaming, enabling real-time voice conversations, audio input/output, and interactive multimodal experiences with your AI agents.

Prerequisites:

- Tutorial 01 (Hello World Agent)
- Tutorial 14 (Streaming with SSE)
- Basic understanding of async/await
- Microphone access for audio examples

What You'll Learn:

- Implementing bidirectional streaming with `StreamingMode.BIDI`
- Using `LiveRequestQueue` for real-time communication
- Configuring audio input/output with speech recognition
- Building voice assistants
- Handling video streaming
- Understanding proactivity and affective dialog
- Live API model selection and compatibility

Time to Complete: 60-75 minutes

Why Live API Matters

Traditional agents are **turn-based** - send message, wait for complete response. The **Live API** enables **real-time, bidirectional** communication:

Turn-Based (Traditional):

```
User speaks → [Complete audio uploaded]
              ↓
Agent thinks → [Processing complete audio]
              ↓
Agent speaks → [Complete response generated]
              ↓
User speaks again...
```

Live API (Bidirectional):

```
User speaks ⟷ Agent hears in real-time
              ⟷ Agent can interrupt
              ⟷ Agent responds while listening
              ⟷ Natural conversation flow
```

Benefits:

- 🎙️ **Real-Time Audio**: Stream audio as you speak
- 🗣️ **Natural Conversations**: Interruptions, turn-taking
- 🎭 **Affective Dialog**: Emotion detection in voice
- 📹 **Video Streaming**: Real-time video analysis
- ⚡ **Low Latency**: Immediate responses
- 🤖 **Proactivity**: Agent can initiate conversation

Getting Started: ADK Web Interface

:::tip RECOMMENDED APPROACH

The **ADK Web Interface** (`adk web`) is the recommended and working method for Live API bidirectional streaming. This approach:

- ✔️ Uses the official `/run_live` WebSocket endpoint
- ✔️ Provides full bidirectional audio streaming
- ✔️ Works out-of-the-box with browser interface
- ✔️ Includes all ADK agent capabilities (tools, state, etc.)

Why not standalone scripts? The `runner.run_live()` method requires an active WebSocket server context with a connected client. Standalone Python scripts don't provide this environment, which is why `adk web` is the official working pattern.

:::

Quick Start with ADK Web

Step 1: Setup

```
cd tutorial_implementation/tutorial15
make setup  # Install dependencies and package
```

Step 2: Configure Environment

```

```

Step 3: Start ADK Web

```
make dev  # Starts web server on http://localhost:8000
```

Step 4: Use in Browser

1. Open http://localhost:8000
2. Select `voice_assistant` from the dropdown
3. Click the **Audio/Microphone button** (🎤)
4. Start your conversation!

How It Works

The ADK web interface provides a `/run_live` WebSocket endpoint that:

```
Browser (Frontend)           ADK Web Server            Gemini Live API
       |                             |                           |
       |---- WebSocket connect ----->|                           |
       |                             |                           |
       |---- LiveRequest (audio) -->|                           |
       |                             |---- process audio ----->|
       |                             |                           |
       |                             |<--- Event (response) ---|
       |<--- Event (audio/text) ----|                           |
       |                             |                           |
```

Key Components:

- **Frontend**: Browser-based UI with microphone/speaker access
- **WebSocket**: `/run_live` endpoint for bidirectional communication
- **Live Request Queue**: Manages message flow between client and agent
- **Concurrent Tasks**: `forward_events()` and `process_messages()` run simultaneously

---

1. Live API Basics

What is Bidirectional Streaming?

BIDI streaming enables **simultaneous** two-way communication between user and agent. Unlike SSE (one-way), BIDI allows:

- User sends data while agent responds
- Agent can respond before user finishes
- Real-time interaction without turn-taking

Source: `google/adk/models/gemini_llm_connection.py`, `google/adk/agents/live_request_queue.py`

Basic Live API Setup

```python
import asyncio
from google.adk.agents import Agent, Runner, RunConfig, StreamingMode, LiveReq
from google.genai import types

# Create agent for live interaction
agent = Agent(
    model='gemini-2.0-flash-live-preview-04-09',  # Live API model (Vertex)
    name='live_assistant',
    instruction='You are a helpful voice assistant. Respond naturally to user
)

# Configure live streaming
run_config = RunConfig(
    streaming_mode=StreamingMode.BIDI,
    speech_config=types.SpeechConfig(
        voice_config=types.VoiceConfig(
            prebuilt_voice_config=types.PrebuiltVoiceConfig(
                voice_name='Puck'  # Available voices: Puck, Charon, Kore, Fen
            )
        )
    )
)

async def live_session():
    """Run live bidirectional session."""

    # Create request queue for live communication
    queue = LiveRequestQueue()

    # Create runner with app or agent
    from google.adk.apps import App
    app = App(name='live_app', root_agent=agent)
    runner = Runner(app=app)

    # Create or get session
    user_id = 'test_user'
    session = await runner.session_service.create_session(
        app_name=app.name,
        user_id=user_id
    )

    # Start live session with correct parameters
    async for event in runner.run_live(
        live_request_queue=queue,
        user_id=user_id,
        session_id=session.id,
```

```python
        run_config=run_config
    ):
        if event.content and event.content.parts:
            # Process agent responses
            for part in event.content.parts:
                if part.text:
                    print(f"Agent: {part.text}")

asyncio.run(live_session())
```

Live API Models

VertexAI API:

```python
# ✔️ Vertex Live API model
agent = Agent(model='gemini-2.0-flash-live-preview-04-09')
```

AI Studio API:

```python
# ✔️ AI Studio Live API model
agent = Agent(model='gemini-live-2.5-flash-preview')
```

Important: Regular Gemini models don't support Live API:

```python
# ❌ These DON'T support Live API
agent = Agent(model='gemini-2.0-flash')  # Regular model
agent = Agent(model='gemini-1.5-flash')  # Older model
```

---

2. LiveRequestQueue: Real-Time Communication

Understanding LiveRequestQueue

`LiveRequestQueue` manages bidirectional communication - sending user input and receiving agent responses simultaneously.

Source: `google/adk/agents/live_request_queue.py`

Sending Text

```python
from google.adk.agents import LiveRequestQueue
from google.genai import types

queue = LiveRequestQueue()

# Send text message using send_content (not send_realtime)
queue.send_content(
    types.Content(
        role='user',
        parts=[types.Part.from_text(text="Hello, how are you?")]
    )
)

# Continue conversation
queue.send_content(
    types.Content(
        role='user',
        parts=[types.Part.from_text(text="Tell me about quantum computing")]
    )
)

# End session
queue.close()
```

Sending Audio

```python
import wave

# Load audio file
with wave.open('audio_input.wav', 'rb') as audio_file:
    audio_data = audio_file.readframes(audio_file.getnframes())

# Send audio to agent using send_realtime (for real-time audio input)
queue.send_realtime(
    blob=types.Blob(
        data=audio_data,
        mime_type='audio/pcm;rate=16000'  # Specify sample rate
    )
)
```

Sending Video

```python
# Send video frame
queue.send_realtime(
    blob=types.Blob(
        data=video_frame_bytes,
        mime_type='video/mp4'
    )
)
```

Queue Management

```python
# Close queue when done
queue.close()

# Queue automatically manages:
# - Buffering
# - Synchronization
# - Backpressure
```

3. Audio Configuration

Speech Recognition (Input)

```
from google.genai import types

run_config = RunConfig(
    streaming_mode=StreamingMode.BIDI,

    # Audio input/output configuration
    speech_config=types.SpeechConfig(
        # Voice output configuration
        voice_config=types.VoiceConfig(
            prebuilt_voice_config=types.PrebuiltVoiceConfig(
                voice_name='Puck'  # Agent's voice
            )
        )
    ),

    # Response format - ONLY ONE modality per session
    response_modalities=['audio']  # For audio responses
    # OR
    # response_modalities=['text']  # For text responses
)
```

Available Voices

```python
# Available prebuilt voices:
voices = [
    'Puck',     # Friendly, conversational
    'Charon',   # Deep, authoritative
    'Kore',     # Warm, professional
    'Fenrir',   # Energetic, dynamic
    'Aoede'     # Calm, soothing
]

# Set voice
run_config = RunConfig(
    streaming_mode=StreamingMode.BIDI,
    speech_config=types.SpeechConfig(
        voice_config=types.VoiceConfig(
            prebuilt_voice_config=types.PrebuiltVoiceConfig(
                voice_name='Charon'  # Choose voice
            )
        )
    )
)
```

Response Modalities

```python
# Text only (use lowercase to avoid Pydantic serialization warnings)
response_modalities=['text']

# Audio only (use lowercase to avoid Pydantic serialization warnings)
response_modalities=['audio']

# CRITICAL: You can only set ONE modality per session
# Native audio models REQUIRE 'audio' modality
# Text-capable models can use 'text' modality
# Setting both ['text', 'audio'] will cause errors
```

4. Building Your Voice Assistant

Project Structure

The Tutorial 15 implementation provides a clean, minimal structure:

```
tutorial_implementation/tutorial15/
├── voice_assistant/
|   ├── __init__.py          # Package exports
|   ├── agent.py             # Core agent & VoiceAssistant class
|   └── audio_utils.py       # AudioPlayer & AudioRecorder utilities
├── tests/                   # Comprehensive test suite
├── Makefile                 # Development commands
├── requirements.txt         # Dependencies
└── pyproject.toml          # Package configuration
```

Core Agent Implementation

The `voice_assistant/agent.py` file defines the root agent that ADK web discovers:

```
"""Voice Assistant Agent for Live API"""

import os
from google.adk.agents import Agent
from google.genai import types

# Environment configuration
LIVE_MODEL = os.getenv(
    "VOICE_ASSISTANT_LIVE_MODEL",
    "gemini-2.0-flash-live-preview-04-09"
)

# Root agent - ADK web will discover this
root_agent = Agent(
    model=LIVE_MODEL,
    name="voice_assistant",
    description="Real-time voice assistant with Live API support",
    instruction="""
You are a helpful voice assistant. Guidelines:

- Respond naturally and conversationally
- Keep responses concise for voice interaction
- Ask clarifying questions when needed
- Be friendly and engaging
- Use casual language appropriate for spoken conversation
    """.strip(),
    generate_content_config=types.GenerateContentConfig(
        temperature=0.8,  # Natural, conversational tone
        max_output_tokens=200  # Concise for voice
    )
)

        ```

That's it! The agent is now discoverable by `adk web`.

Using the Voice Assistant

Once you've created the agent and run `make dev`, the ADK web server:

1. **Discovers** the `root_agent` from `voice_assistant/agent.py`
2. **Creates** a `/run_live` WebSocket endpoint
3. **Handles** bidirectional audio streaming automatically
4. **Manages** the LiveRequestQueue and concurrent event processing

In the browser:
```
```

```
- Select `voice_assistant` from the dropdown
- Click the Audio/Microphone button
- Start speaking or typing
- The agent responds in real-time with audio output

###AudioUtilities (Optional)

For programmatic audio handling, `voice_assistant/audio_utils.py` provides:

```python
from voice_assistant.audio_utils import AudioPlayer, AudioRecorder

# Play PCM audio
player = AudioPlayer()
player.play_pcm_bytes(audio_data)
player.save_to_wav(audio_data, "output.wav")
player.close()

# Record from microphone
recorder = AudioRecorder()
audio_data = recorder.record(duration_seconds=5)
recorder.save_to_wav(audio_data, "input.wav")
recorder.close()
```

Configuration Options

Environment Variables:

```
# Model selection

# Vertex AI configuration
```

Voice Selection (modify agent.py):
```

```python
# Add speech_config to run_config in VoiceAssistant class
run_config = RunConfig(
    streaming_mode=StreamingMode.BIDI,
    speech_config=types.SpeechConfig(
        voice_config=types.VoiceConfig(
            prebuilt_voice_config=types.PrebuiltVoiceConfig(
                voice_name='Charon'  # Options: Puck, Charon, Kore, Fenrir, Ao
            )
        )
    )
)
```

Testing

Run the comprehensive test suite:

```
make test
```

Tests verify:
- ✅ Agent configuration
- ✅ VoiceAssistant class functionality
- ✅ Package structure and imports
- ✅ Audio utilities availability

---

5. Advanced Live API Features

Proactivity

Allow agent to initiate conversation:

```python
from google.genai import types

run_config = RunConfig(
    streaming_mode=StreamingMode.BIDI,

    # Enable proactive responses (requires v1alpha API)
    # Note: Proactive audio only supported by native audio models
    proactivity=types.ProactivityConfig(
        proactive_audio=True
    ),

    speech_config=types.SpeechConfig(
        voice_config=types.VoiceConfig(
            prebuilt_voice_config=types.PrebuiltVoiceConfig(
                voice_name='Puck'
            )
        )
    )
)

# Agent can now speak without waiting for user input
# Useful for: notifications, reminders, suggestions
```

Affective Dialog (Emotion Detection)

Detect user emotions from voice:

```
run_config = RunConfig(
    streaming_mode=StreamingMode.BIDI,

    # Enable emotion detection
    enable_affective_dialog=True,

    speech_config=types.SpeechConfig(
        voice_config=types.VoiceConfig(
            prebuilt_voice_config=types.PrebuiltVoiceConfig(
                voice_name='Kore'  # Empathetic voice
            )
        )
    )
)

# Agent receives emotion signals:
# - Happy, Sad, Angry, Neutral, etc.
# - Can adjust response tone accordingly
```

Video Streaming

Stream video for real-time analysis:

```python
import cv2

# Capture video
cap = cv2.VideoCapture(0)

queue = LiveRequestQueue()

while True:
    ret, frame = cap.read()

    if not ret:
        break

    # Convert frame to bytes
    _, buffer = cv2.imencode('.jpg', frame)
    frame_bytes = buffer.tobytes()

    # Send frame to agent
    queue.send_realtime(
        blob=types.Blob(
            data=frame_bytes,
            mime_type='image/jpeg'
        )
    )

    await asyncio.sleep(0.1)  # ~10 FPS

queue.send_end()

# Agent can analyze video in real-time
# Use cases: gesture recognition, object detection, surveillance
```

6. Multi-Agent Live Sessions

Combine multiple agents in live conversation:

```
"""
Multi-agent voice conversation.
"""

from google.adk.agents import Agent, Runner, RunConfig, StreamingMode, LiveReq
from google.genai import types

# Create specialized agents
greeter = Agent(
    model='gemini-2.0-flash-live-preview-04-09',
    name='greeter',
    instruction='Greet users warmly and ask how you can help.'
)

expert = Agent(
    model='gemini-2.0-flash-live-preview-04-09',
    name='expert',
    instruction='Provide detailed expert answers to questions.'
)

# Orchestrator agent
orchestrator = Agent(
    model='gemini-2.0-flash-live-preview-04-09',
    name='orchestrator',
    instruction="""
You coordinate between multiple agents:
- Use 'greeter' for initial contact
- Use 'expert' for detailed questions
- Ensure smooth conversation flow
    """,
    sub_agents=[greeter, expert],
    flow='sequential'
)

run_config = RunConfig(
    streaming_mode=StreamingMode.BIDI,
    speech_config=types.SpeechConfig(
        voice_config=types.VoiceConfig(
            prebuilt_voice_config=types.PrebuiltVoiceConfig(
                voice_name='Puck'
            )
        )
    )
)

async def multi_agent_voice():
```

```python
    """Run multi-agent voice session."""

    queue = LiveRequestQueue()

    # Setup app and runner
    from google.adk.apps import App
    app = App(name='multi_agent_voice', root_agent=orchestrator)
    runner = Runner(app=app)

    # Create session
    user_id = 'multi_agent_user'
    session = await runner.session_service.create_session(
        app_name=app.name,
        user_id=user_id
    )

    # User speaks (use send_content for text)
    queue.send_content(
        types.Content(
            role='user',
            parts=[types.Part.from_text(
                text="Hello, I have a question about quantum computing"
            )]
        )
    )
    queue.close()

    # Orchestrator coordinates agents
    async for event in runner.run_live(
        live_request_queue=queue,
        user_id=user_id,
        session_id=session.id,
        run_config=run_config
    ):
        if event.content and event.content.parts:
            for part in event.content.parts:
                if part.text:
                    print(f"{event.author}: {part.text}")

asyncio.run(multi_agent_voice())
```

7. Best Practices

✔️ DO: Use Live API Models

```
# ✔️ Good - Live API models
agent = Agent(model='gemini-2.0-flash-live-preview-04-09')  # Vertex
agent = Agent(model='gemini-live-2.5-flash-preview')  # AI Studio

# ❌ Bad - Regular models don't support Live API
agent = Agent(model='gemini-2.0-flash')
agent = Agent(model='gemini-1.5-flash')
```

✔️ DO: Keep Voice Responses Concise

```
# ✔️ Good - Concise for voice
agent = Agent(
    model='gemini-2.0-flash-live-preview-04-09',
    instruction='Keep responses brief and conversational for voice interaction
    generate_content_config=types.GenerateContentConfig(
        max_output_tokens=150
    )
)

# ❌ Bad - Too verbose for voice
agent = Agent(
    model='gemini-2.0-flash-live-preview-04-09',
    generate_content_config=types.GenerateContentConfig(
        max_output_tokens=4096  # Too long for voice
    )
)
```

✔️ DO: Handle Audio Formats Properly

```python
# ✔️ Good - Correct audio format with sample rate
queue.send_realtime(
    blob=types.Blob(
        data=audio_data,
        mime_type='audio/pcm;rate=16000'  # Specify sample rate
    )
)

# ❌ Bad - Wrong format or missing rate
queue.send_realtime(
    blob=types.Blob(
        data=audio_data,
        mime_type='text/plain'  # Wrong type
    )
)
```

✔️ DO: Always Close Queue

```python
# ✔️ Good - Properly close queue
queue = LiveRequestQueue()

try:
    queue.send_content(types.Content(
        role='user',
        parts=[types.Part.from_text(text="Hello")]
    ))
    # ... process responses
finally:
    queue.close()  # Always close

# ❌ Bad - Forgot to close
queue = LiveRequestQueue()
queue.send_content(types.Content(
    role='user',
    parts=[types.Part.from_text(text="Hello")]
))
# Queue left open
```

✅ DO: Use Appropriate Voices

```python
# ✅ Good - Voice matches use case
customer_service = Agent(
    model='gemini-2.0-flash-live-preview-04-09',
    instruction='Helpful customer service agent'
)

run_config = RunConfig(
    streaming_mode=StreamingMode.BIDI,
    speech_config=types.SpeechConfig(
        voice_config=types.VoiceConfig(
            prebuilt_voice_config=types.PrebuiltVoiceConfig(
                voice_name='Kore'  # Warm, professional
            )
        )
    )
)
```

8. Troubleshooting

Error: "Model doesn't support Live API"

Problem: Using non-Live API model

Solution:

```python
# ❌ Wrong model
agent = Agent(model='gemini-2.0-flash')

# ✅ Use Live API model
agent = Agent(model='gemini-2.0-flash-live-preview-04-09')  # Vertex
# Or
agent = Agent(model='gemini-live-2.5-flash-preview')  # AI Studio
```

Issue: "No audio in response"

Problem: Audio not configured properly

Solutions:

1. **Set response modalities**:

```
run_config = RunConfig(
    streaming_mode=StreamingMode.BIDI,
    response_modalities=['TEXT', 'AUDIO'],  # Include AUDIO
    speech_config=types.SpeechConfig(...)
)
```

1. **Configure voice**:

```
speech_config=types.SpeechConfig(
    voice_config=types.VoiceConfig(
        prebuilt_voice_config=types.PrebuiltVoiceConfig(
            voice_name='Puck'  # Must set voice
        )
    )
)
```

Issue: "Queue timeout"

Problem: Queue not properly closed

Solution:

```
# ✔️ Always close() the queue
queue = LiveRequestQueue()
queue.send_content(types.Content(
    role='user',
    parts=[types.Part.from_text(text="Hello")]
))
queue.close()  # Important!
```

Summary

:::tip IMPLEMENTATION RECOMMENDATION

For Production Live API Applications: Use the `adk web` interface as demonstrated in this tutorial. The `/run_live` WebSocket endpoint is the official, tested pattern for bidirectional audio streaming.

Why ADK Web Works:
- Active WebSocket connection between browser and server
- Concurrent task management (`forward_events()` + `process_messages()`)
- Proper LiveRequestQueue handling
- Full ADK agent capabilities (tools, state, memory)

Alternative: For applications that need direct API access without the ADK framework, use `google.genai.Client.aio.live.connect()` directly (bypasses ADK Runner).

:::

You've mastered the Live API for real-time voice interactions:

Key Takeaways:

- ✅ `StreamingMode.BIDI` enables bidirectional streaming
- ✅ `LiveRequestQueue` manages real-time communication
- ✅ Audio input/output with `speech_config`
- ✅ Multiple voices available (Puck, Charon, Kore, etc.)
- ✅ Proactivity for agent-initiated conversation
- ✅ Affective dialog for emotion detection
- ✅ Video streaming support
- ✅ Live API models: `gemini-2.0-flash-live-preview-04-09` (Vertex), `gemini-live-2.5-flash-preview` (AI Studio)

Production Checklist:

- [ ] Using Live API compatible model
- [ ] `StreamingMode.BIDI` configured
- [ ] Speech config with voice selection
- [ ] Audio format properly set (audio/pcm;rate=16000)
- [ ] Queue properly closed with `close()`
- [ ] Concise responses for voice (max_output_tokens=150-200)
- [ ] Error handling for audio/network issues
- [ ] Testing with actual audio devices
- [ ] Only ONE response modality per session (TEXT or AUDIO, not both)

- [ ] Correct run_live() parameters (live_request_queue, user_id, session_id)

Next Steps:

- **Tutorial 16**: Learn MCP Integration for extended tool ecosystem
- **Tutorial 17**: Implement Agent-to-Agent (A2A) communication
- **Tutorial 18**: Master Events & Observability

Resources:

- [Live API Documentation](https://cloud.google.com/vertex-ai/generative-ai/docs/model-reference/gemini-live) (https://cloud.google.com/vertex-ai/generative-ai/docs/model-reference/gemini-live)
- [Audio Configuration Guide](https://cloud.google.com/vertex-ai/generative-ai/docs/speech) (https://cloud.google.com/vertex-ai/generative-ai/docs/speech)
- [Sample: live_bidi_streaming_single_agent](https://github.com/google/adk-python/tree/main/contributing/samples/live_bidi_streaming_single_agent/) (https://github.com/google/adk-python/tree/main/contributing/samples/live_bidi_streaming_single_agent/)

🎉 **Tutorial 15 Complete!** You now know how to build real-time voice assistants with the Live API. Continue to Tutorial 16 to learn about MCP integration for extended capabilities.

Generated on 2025-10-19 17:56:49 from 15_live_api_audio.md

Source: Google ADK Training Hub