AKOOL

Streaming Avatar

Streaming Avatar Integration Guide

Learn how to integrate streaming avatars using Agora, LiveKit, or TRTC SDK

🎉 **New Feature Available!** Video interaction with streaming avatars is now live! You can now enable two-way video communication with your avatars, including camera switching capabilities and video quality controls. Check out the **Video Interaction** section for implementation details.

Overview

The Streaming Avatar feature allows you to create interactive, real-time avatar experiences in your application. This guide provides a comprehensive walkthrough of integrating streaming avatars using **Agora SDK**, **LiveKit SDK**, or **TRTC SDK**, including:

- Setting up real-time communication channels
- Handling avatar interactions and responses
- Managing audio streams
- Implementing cleanup procedures
- Optional LLM service integration

Choose between three reliable WebRTC-based SDKs for low-latency streaming with our avatar service. The code examples in this guide use **synchronized tabs** - select your preferred provider (Agora, LiveKit, or TRTC) in any code block, and all code examples on the page will automatically switch to match your selection.

Prerequisites

1. Install the SDK

Agora LiveKit TRTC

```
npm install agora-rtc-sdk-ng
# or
yarn add agora-rtc-sdk-ng
```

2. Import the required dependencies

Agora LiveKit TRTC

```
import AgoraRTC, { IAgoraRTCClient } from "agora-rtc-sdk-ng";
```

3. Understanding Data Channel Limitations

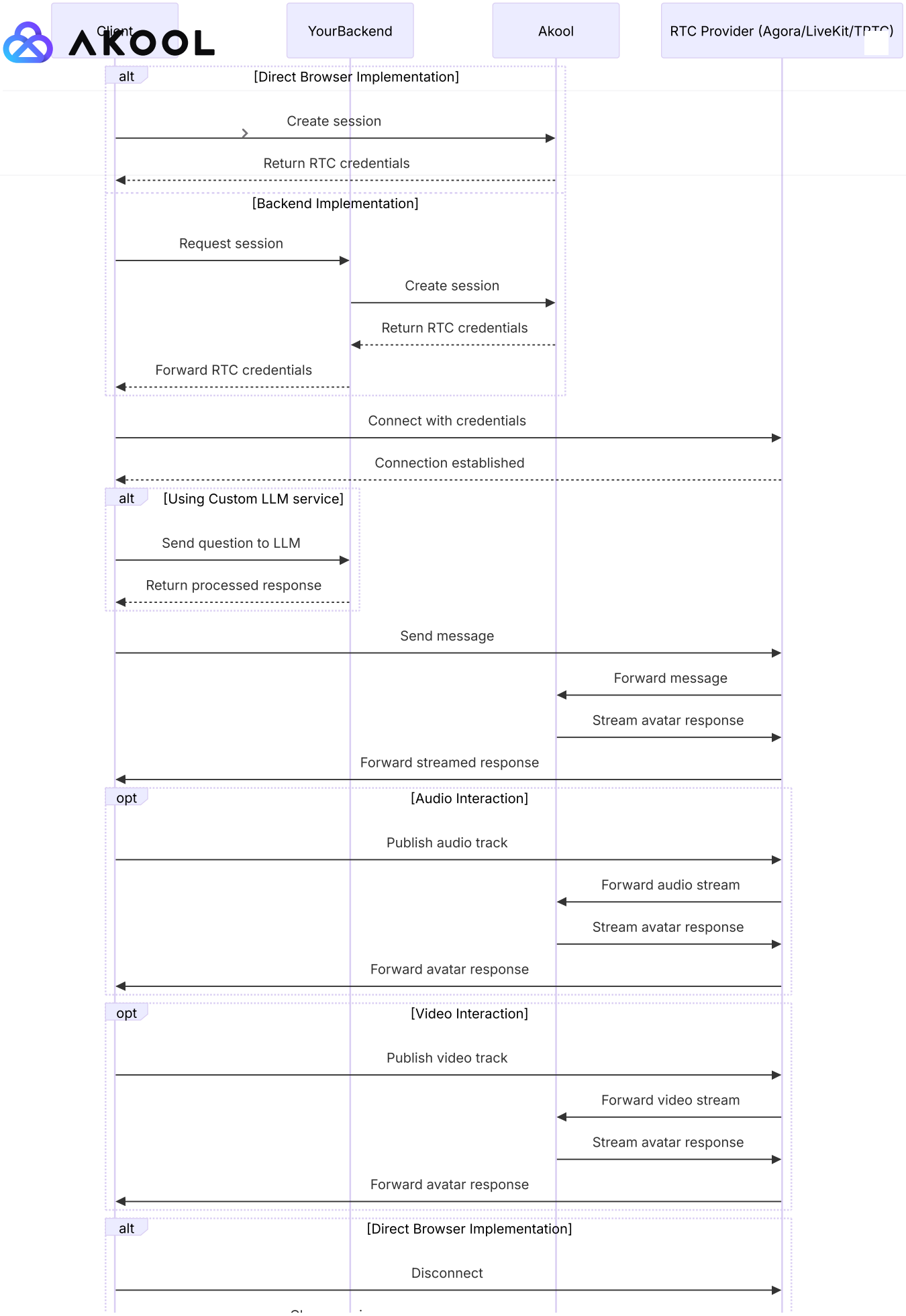
Agora SDK - Hidden API and Limitations

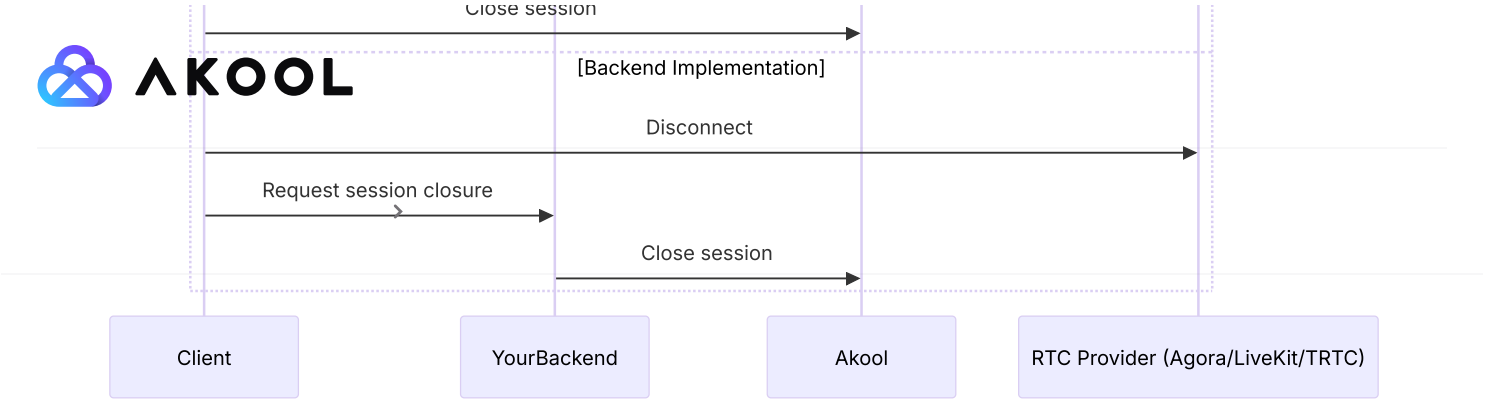
LiveKit SDK - Data Channel Limitations

TRTC SDK - Data Channel Limitations

Integration Flow

The integration follows the same pattern regardless of which SDK you choose (Agora, LiveKit, or TRTC):


Client
YourBackend
Akool
RTC Provider (Agora/LiveKit/TRTC)
alt
[Direct Browser Implementation]
Create session
Return RTC credentials
[Backend Implementation]
Request session
Create session
Return RTC credentials
Forward RTC credentials
Connect with credentials
Connection established
alt
[Using Custom LLM service]
Send question to LLM
Return processed response
Send message
Forward message
Stream avatar response
Forward streamed response
opt
[Audio Interaction]
Publish audio track
Forward audio stream
Stream avatar response
Forward avatar response
opt
[Video Interaction]
Publish video track
Forward video stream
Stream avatar response
Forward avatar response
alt
[Direct Browser Implementation]
Disconnect



Key Implementation Steps

1. Create a Live Avatar Session

⚠ **Security Recommendation**: We strongly recommend implementing session management through your backend server rather than directly in the browser. This approach:

- Protects your AKool API token from exposure
- Allows for proper request validation and rate limiting
- Enables usage tracking and monitoring
- Provides better control over session lifecycle
- Prevents unauthorized access to the API

First, create a session to obtain Agora credentials. While both browser and backend implementations are possible, the backend approach is recommended for security:

```
// Recommended: Backend Implementation
async function createSessionFromBackend(): Promise<Session> {
  // Your backend endpoint that securely wraps the AKool API
  const response = await fetch('https://your-backend.com/api/avatar/crea
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      avatarId: "dvp_Tristan_cloth2_1080P",
      duration: 600,
    })
  });

  if (!response.ok) {
    throw new Error('Failed to create session through backend');
  }

  return response.json();
}

// Not Recommended: Direct Browser Implementation
// Only use this for development/testing purposes
async function createSessionInBrowser(): Promise<Session> {
  const response = await fetch('https://openapi.akool.com/api/open/v4/li
    method: 'POST',
    headers: {
      'x-api-key':'{{API Key}}', // Security risk: Token exposed in brow
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      avatar_id: "dvp_Tristan_cloth2_1080P",
      duration: 600,
    })
  });
```

```
  if (!response.ok) {
    throw new Error(`Failed to create session: ${response.status} ${resp
  }

  const res = await response.json();
  return res.data;
}
```

2. Initialize the Client/Room

Create and configure the client or room:

Agora LiveKit TRTC

```
async function initializeAgoraClient(credentials) {
  const client = AgoraRTC.createClient({
    mode: 'rtc',
    codec: 'vp8'
  });

  try {
    await client.join(
      credentials.agora_app_id,
      credentials.agora_channel,
      credentials.agora_token,
      credentials.agora_uid
    );

    return client;
  } catch (error) {
    console.error('Error joining channel:', error);
    throw error;
  }
}
```

3. Subscribe to Audio and Video Stream

Subscribe to the audio and video stream of the avatar:

Agora LiveKit TRTC

```
async function subscribeToAvatarStream(client: IAgoraRTCClient) {
  const onUserPublish = async (user: IAgoraRTCRemoteUser, mediaType: 'v
    const remoteTrack = await client.subscribe(user, mediaType);
    remoteTrack.play();
  };

  const onUserUnpublish = async (user: IAgoraRTCRemoteUser, mediaType:
    await client.unsubscribe(user, mediaType);
  };

  client.on('user-published', onUserPublish);
  client.on('user-unpublished', onUserUnpublish);
}
```

4. Set Up Message Handling

Configure message listeners to handle avatar responses:

Agora LiveKit TRTC

```
function setupMessageHandlers(client: IAgoraRTCClient) {
  let answer = '';
  client.on('stream-message', (uid, message) => {
    try {
      const parsedMessage = JSON.parse(message);

      if (parsedMessage.type === 'chat') {
        const payload = parsedMessage.pld;

        if (payload.from === 'bot') {
          if (!payload.fin) {
            answer += payload.text;
          } else {
            console.log('Avatar response:', answer);
            answer = '';
          }
        } else if (payload.from === 'user') {
          console.log('User message:', payload.text);
        }
      } else if (parsedMessage.type === 'command') {
        if (parsedMessage.pld.code !== 1000) {
          console.error('Command failed:', parsedMessage.pld.msg);
        }
      }
    } catch (error) {
      console.error('Error parsing message:', error);
    }
  });
}
```

5. Send Messages to Avatar

Implement functions to interact with the avatar:

AKOOL

Agora LiveKit TRTC

```
async function sendMessageToAvatar(client: IAgoraRTCClient, question: s
  const message = {
    v: 2,
    type: "chat",
    mid: `msg-${Date.now()}`,
    idx: 0,
    fin: true,
    pld: {
      text: question,
    }
  };

  try {
    await client.sendStreamMessage(JSON.stringify(message), false);
  } catch (error) {
    console.error('Error sending message:', error);
    throw error;
  }
}
```

Agora: Handling Large Messages with Chunking

LiveKit Note: Unlike Agora's 1KB limit, LiveKit supports messages up to 15 KiB in reliable mode, so chunking is generally not needed for typical conversational messages. For very large messages (over 15 KiB), you would need to implement similar chunking logic.

TRTC Note: TRTC has the same 1KB message size limit as Agora, so you would need to implement similar chunking logic for large messages. The chunking

approach shown above for Agora can be adapted for TRTC by replacing `client.sendStreamMessage()` with `client.sendCustomMessage()`.

6. Control Avatar Parameters

Implement functions to control avatar settings:

Agora LiveKit TRTC

```
async function setAvatarParams(client: IAgoraRTCClient, params: {
  vid?: string;
  lang?: string;
  mode?: number;
  bgurl?: string;
}) {
  const message = {
    v: 2,
    type: 'command',
    mid: `msg-${Date.now()}`,
    pld: {
      cmd: 'set-params',
      data: params
    }
  };

  await client.sendStreamMessage(JSON.stringify(message), false);
}

async function interruptAvatar(client: IAgoraRTCClient) {
  const message = {
    v: 2,
    type: 'command',
    mid: `msg-${Date.now()}`,
    pld: {
      cmd: 'interrupt'
    }
  };

  await client.sendStreamMessage(JSON.stringify(message), false);
}
```

7. Audio Interaction With The Avatar

To enable audio interaction with the avatar, you'll need to publish your local audio stream:

Agora LiveKit TRTC

AKOOL

```
async function publishAudio(client: IAgoraRTCClient) {
  // Create a microphone audio track
  const audioTrack = await AgoraRTC.createMicrophoneAudioTrack();

  try {
    // Publish the audio track to the channel
    await client.publish(audioTrack);
    console.log("Audio publishing successful");

    return audioTrack;
  } catch (error) {
    console.error("Error publishing audio:", error);
    throw error;
  }
}

// Example usage with audio controls
async function setupAudioInteraction(client: IAgoraRTCClient) {
  let audioTrack;

  // Start audio
  async function startAudio() {
    try {
      audioTrack = await publishAudio(client);
    } catch (error) {
      console.error("Failed to start audio:", error);
    }
  }

  // Stop audio
  async function stopAudio() {
    if (audioTrack) {
      // Stop and close the audio track
      audioTrack.stop();
```

```
      audioTrack.close();
      await client.unpublish(audioTrack);
      audioTrack = null;
    }
  }

  // Mute/unmute audio
  function toggleAudio(muted: boolean) {
    if (audioTrack) {
      if (muted) {
        audioTrack.setEnabled(false);
      } else {
        audioTrack.setEnabled(true);
      }
    }
  }

  return {
    startAudio,
    stopAudio,
    toggleAudio
  };
}
```

Now you can integrate audio controls into your application:

Agora LiveKit TRTC

```
async function initializeWithAudio() {
  try {
    // Initialize avatar
    const client = await initializeStreamingAvatar();

    // Setup audio controls
    const audioControls = await setupAudioInteraction(client);

    // Start audio when needed
    await audioControls.startAudio();

    // Example of muting/unmuting
    audioControls.toggleAudio(true); // mute
    audioControls.toggleAudio(false); // unmute

    // Stop audio when done
    await audioControls.stopAudio();

  } catch (error) {
    console.error("Error initializing with audio:", error);
  }
}
```

Additional Resources:

- Agora Web SDK Documentation
- LiveKit Web SDK Documentation

8. Integrating your own LLM service (optional)

You can integrate your own LLM service to process messages before sending them to the avatar. Here's how to do it:

```
// Define the LLM service response interface
interface LLMResponse {
  answer: string;
}

// Set the avatar to retelling mode
await setAvatarParams(client, {
  mode: 1,
});

// Create a wrapper for your LLM service
async function processWithLLM(question: string): Promise<LLMResponse> {
  try {
    const response = await fetch('YOUR_LLM_SERVICE_ENDPOINT', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({
        question,
      })
    });

    if (!response.ok) {
      throw new Error('LLM service request failed');
    }

    return await response.json();
  } catch (error) {
    console.error('Error processing with LLM:', error);
    throw error;
  }
}
```

Agora LiveKit TRTC

```
async function sendMessageToAvatarWithLLM(
  client: IAgoraRTCClient,
  question: string
) {
  try {
    // Process the question with your LLM service
    const llmResponse = await processWithLLM(question);

    // Prepare the message with LLM response
    const message = {
      v: 2,
      type: "chat",
      mid: `msg-${Date.now()}`,
      idx: 0,
      fin: true,
      pld: {
        text: llmResponse.answer // Use the LLM-processed response
      }
    };

    // Send the processed message to the avatar
    await client.sendStreamMessage(JSON.stringify(message), false);

  } catch (error) {
    console.error('Error in LLM-enhanced message sending:', error);
    throw error;
  }
}
```

Remember to:

1. Implement proper rate limiting for your LLM service

2. Handle token limits appropriately
3. Implement retry logic for failed LLM requests
4. Consider implementing streaming responses if your LLM service supports it
5. Cache common responses when appropriate

9. Cleanup

Cleanup can also be performed either directly or through your backend:

Agora LiveKit TRTC

```
// Browser Implementation
async function cleanupInBrowser(client: IAgoraRTCClient, sessionId: str
  await fetch('https://openapi.akool.com/api/open/v4/liveAvatar/session
    method: 'POST',
    headers: {
      'x-api-key':'{{API Key}}'
    },
    body: JSON.stringify({
      id: sessionId
    })
  });

  await performClientCleanup(client);
}

// Backend Implementation
async function cleanupFromBackend(client: IAgoraRTCClient, sessionId: s
  await fetch('https://your-backend.com/api/avatar/close-session', {
    method: 'POST',
    body: JSON.stringify({
      sessionId
    })
  });

  await performClientCleanup(client);
}

// Shared cleanup logic
async function performClientCleanup(client: IAgoraRTCClient) {
  // Remove event listeners
  client.removeAllListeners('user-published');
  client.removeAllListeners('user-unpublished');
  client.removeAllListeners('stream-message');
```

```
  // Stop audio/video and unpublish if they're still running
  if (audioControls) {
    await audioControls.stopAudio();
  }
  if (videoControls) {
    await videoControls.stopVideo();
  }

  // Leave the Agora channel
  await client.leave();
}
```

When implementing through your backend, make sure to:

- Securely store your AKool API token
- Implement proper authentication and rate limiting
- Handle errors appropriately
- Consider implementing session management and monitoring

10. Putting It All Together

Here's how to use all the components together:

Agora LiveKit TRTC

```
async function initializeStreamingAvatar() {
  let client;
  let session;
  try {
    // Create session and get credentials
    session = await createSession();
    const { credentials } = session;

    // Initialize Agora client
    client = await initializeAgoraClient(credentials);

    // Subscribe to the audio and video stream of the avatar
    await subscribeToAvatarStream(client);

    // Set up message handlers
    setupMessageHandlers(client);

    // Example usage
    await sendMessageToAvatar(client, "Hello!");

    // Or use your own LLM service
    await sendMessageToAvatarWithLLM(client, "Hello!");

    // Example of voice interaction
    await interruptAvatar(client);

    // Example of Audio Interaction With The Avatar
    await setupAudioInteraction(client);

    // Example of changing avatar parameters
    await setAvatarParams(client, {
      lang: "en",
      vid: "new_voice_id"
    });
```

```
    return client;
  } catch (error) {
    console.error('Error initializing streaming avatar:', error);
    if (client && session) {
      await cleanup(client, session._id);
    }
    throw error;
  }
}
```

Additional Resources

Agora SDK Resources

- Agora Web SDK Documentation
- Agora Web SDK API Reference
- Agora Data Stream Error Codes

LiveKit SDK Resources

- LiveKit Web SDK Documentation
- LiveKit Web SDK API Reference
- LiveKit Data Channel Documentation

TRTC SDK Resources

- TRTC Web SDK Documentation
- TRTC Web SDK API Reference
- TRTC Custom Message Documentation

AKool API Resources

AKool OpenAPI Error Codes

Get Session List

Streaming Avatar SDK Quick Start