实时音视频

AI 实时对话

腾讯云

【版权声明】

【商标声明】

【服务声明】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

AI 实时对话
功能介绍

最近更新时间：2025-03-04 08:52:32

功能概述

腾讯 AI 实时对话解决方案，支持客户灵活接入多家 AI 大模型服务，实现 AI 与用户之间的实时音视频互动，打造符合业务场景的 AI 实时对话能力。基于腾讯 RTC 全球低延迟传输，语音对话延迟低至1s，对话效果自然拟真，接入便捷，开箱即用。

计费说明

该功能的使用费用包含三个模块：音频通话费用、AI实时对话服务费用和语音转文字服务费用。
具体费用详情请参见 AI 实时对话计费说明 。

Demo 体验

Web 端	Android 端	iOS 端

AI 实时对话 Demo	Android 端 Demo	iOS 端 Demo

无代码快速跑通 AI 对话

腾讯 RTC 提供了便捷的测试验证平台，支持您对接第三方的 STT、LLM 和 TTS 能力，快速配置，无代码快速跑通对话式 AI 功能进行验证测试，实现高效开发和部署。
配置地址：实时音视频控制台 > 快速跑通 AI 实时对话 。

应用场景

应用场景	说明
在线教育	在线教育场景中，实时互动和反馈是提升学习效果的关键。依托 AI 实时对话，平台可以创建虚拟教学助手，在课内课外提供全时的智能教学辅助。 • 课内，学生可以在老师讲课的同时，随时向虚拟教学助手提问，获得补充讲解，更充分地理解学习要点。 • 课外，虚拟教学助手可以根据不同学生的进度和需求，提供个性化的辅导建议和学习资源，并针对学生的作业、提问提供响应式反馈，以更自然亲和的方式陪伴学生。相比大段的文字解析，对话式的讲解可以更有效地引导学生，便于学生理解。
社交娱乐	在社交娱乐场景中，结合实时互动能力的 AI 实时对话能够精准理解用户意图并和用户进行语音互动，为用户带来更真实和个性化的社交娱乐体验。相比文字，AI 实时对话提供的虚拟陪伴服务能够通过语音与用户进行自然沟通，提供更为丰富且真实的情感价值。在线剧本杀、狼人杀等互动游戏中，AI 实时对话也可以扮演主持或 NPC 角色，与玩家进行动态 AI 实时对话并推动情节发展，让玩家享受沉浸式的游戏体验。
呼叫中心	在线客服、AI 销售顾问、智能外呼等场景都可以通过 AI 实时对话来提供更丰富、实时的客户服务体验，这样不仅可以有效降低运营成本，还能够显著提升服务效率，全天候为客户提供更快捷的服务支持。
高效办公	通过 AI 实时对话，用户可以使用语音，命令和控制应用程序，减少手动输入，使日常工作变得更轻松、高效。相比文字交互，对话式交互可以拓展各类办公助手的使用场景，无需在终端设备旁也能通过语音快速交流，完成工作。
医疗辅助	依托 AI 实时对话，远程诊断、医疗咨询等场景中，患者可以通过语音咨询提问，获取实时且个性化的建议，更接近真实场景问诊体验，这样可以消除用户的不信任感，大幅减轻患者焦虑。

> **说明：**
> 请查看行业实践教程：情感陪伴 、智能客服 。

功能优势

优势	说明
超低延迟的 AI 实时对话	在实时人工智能交互场景中，LLM 及时接收和处理用户的音视频数据至关重要。腾讯 RTC 的超低延迟通信确保了全球范围内音视频传输的端到端延迟低于300ms，同时将对话延迟保持在1000ms以下，媲美人类对话反应速度，让用户享受到流畅自然的互动体验，提升客户满意度。
接入简单，高效上线	提供无代码快速跑通平台，仅需10min，可预先快速验证解决方案。正式集成提供含 UI 的对接方式以及完整的 SDK 和 API 文档，简化开发流程，集成可在1 - 2天内完成，比传统方案节省1个月以上的开发工作，助力企业快速实现产品智能化升级，抢占市场先机。
自然拟真的对话	我们支持多种语言的输入，包括英语、西班牙语、日语、韩语、中文等130种国际语言。支持为最多三种指定语言提供模糊识别（不包括方言），确保了识别的高精度和适应性。结合自研 VAD 技术实现 AI 对话智能语义打断，更好的适应人类对话节奏与响应速度，带来超拟人的对话互动体验。
服务模型集成灵活性	与各种 LLM 和 TTS 模型无缝集成：我们提供了集成通道，允许用户轻松连接第三方 LLM 和 TTS 模型。用户只需要配置他们的 LLM 和 TTS 服务的账户凭证，就可以将它们无缝集成到我们的解决方案中。这促进了个性化和复杂的 AI 响应，增强了整体对话体验。
跨端兼容性	支持多个平台，包括 iOS、Android、Windows、macOS、Web、Flutter、Electron 和 React Native 等，兼容超过20,000种设备模型。
领先的音频处理	支持服务器端和客户端 AI 降噪，声纹识别和 3A 回声消除功能，可以根据不同的AI聊天模式进行定制，提高语音识别精度和 AI 通话质量，实现各种场景下的精准高清 AI 对话。

无代码快速跑通 AI 实时对话

最近更新时间：2025-02-13 11:51:22

腾讯 TRTC 提供了便捷的测试验证平台，支持您对接第三方的 LLM 和 TTS 能力，快速配置，无代码快速跑通 AI 实时对话功能进行验证测试，实现高效开发和部署。本文将介绍如何使用该工具。

登录控制台

登录控制台配置地址：实时音视频控制台 > 快速跑通AI实时对话 。

资源配置

快速跑通 AI 实时对话支持对接多家 LLM、TTS 厂商，单击**开始使用**即可在此快速配置、测试并集成AI实时对话服务，实现高效开发与部署。

> **说明：**
> - 单击 体验 AI 对话 Demo 体验 AI 对话的能力。
> - 使用快速跑通 AI 实时对话测试工具将产生用量，计费详情请参见 AI 实时对话计费说明 。

STEP 1：基础配置

1. 选择应用

在**基础配置**页，您首先需要选择需要测试的应用。您可通过**领取体验版**或 **购买 AI 智能识别套餐包** 开通应用的 STT 服务来使用 STT 语音识别功能。

示例：

当前应用未开通 STT 服务，可点击领取体验版，在弹框中免费领取。

2. STT 配置项

我们提供了默认参数，您可以根据需求对 STT 配置项进行微调。配置完成后单击下一步。

- **填写机器人欢迎语。**
- **选择打断模式。可基于特定规则自动打断机器人讲话，该模式下同时支持自动打断和手动打断，具体描述可参见 打断模式介绍 。**
- **选择打断时长。**

STEP 2：STT 语音识别配置

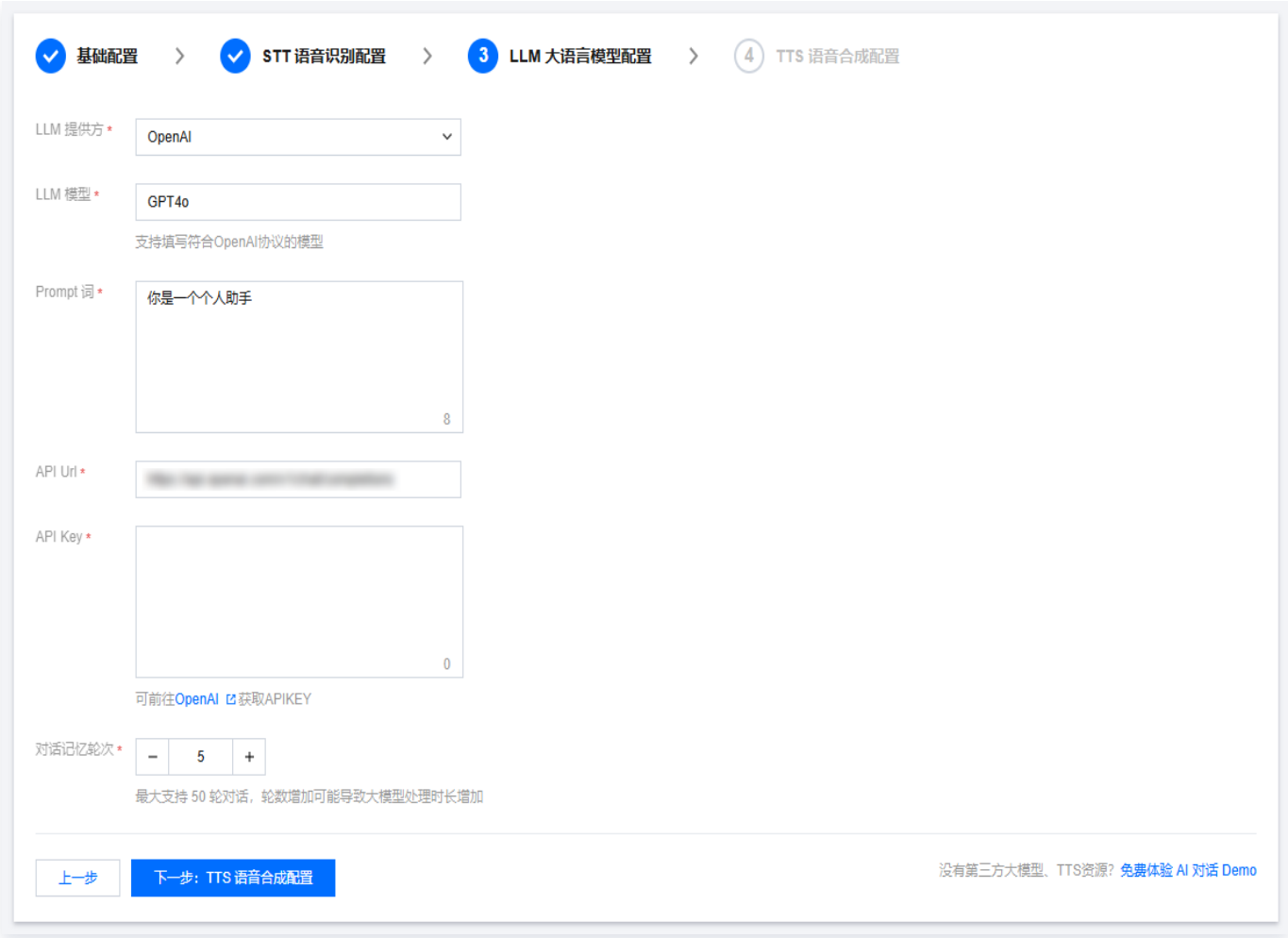
STT 语音识别配置模型仅可选择腾讯 STT，支持选择多种语言，VAD 时长可调整，设置范围为240 – 2000，更小的值会让语音识别分句更快。配置完成后单击下一步。

STEP 3：LLM 大语言模型配置

在本页面需要您自行申请第三方资源进行配置：LLM 提供方支持 OpenAI、Deepseek、Minimax、混元、Coze 和 Dify。

OpenAI

LLM 提供方选择 OpenAI 时，此处支持任何符合 OpenAI 标准协议的 LLM 模型，您可以自行调整下方的模型、Url 和 Key 进行调用。当您选择使用 OpenAI 提供的模型时，API Key 可前往 OpenAI 获取。



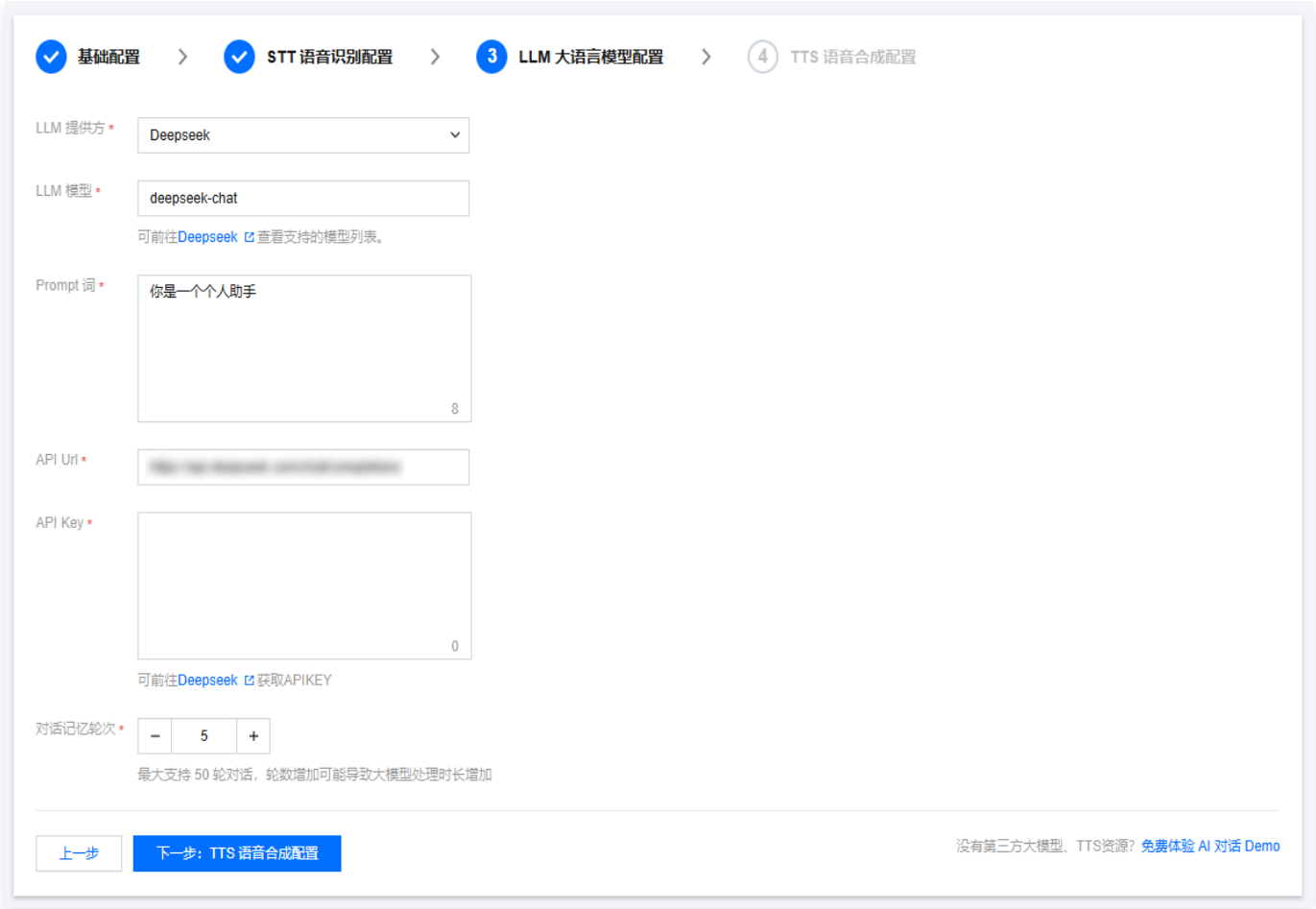
Deepseek

LLM 提供方选择 Deepseek 时，其他配置我们提供了默认参数，您根据需求进行微调；LLM 模型可前往 Deepseek 查看支持的模型列表，API Key 可前往 Deepseek 获取。
除了Deepseek官方，您还可以选择调用第三方云平台部署的 Deepseek 服务：

- **腾讯云 TI 平台**

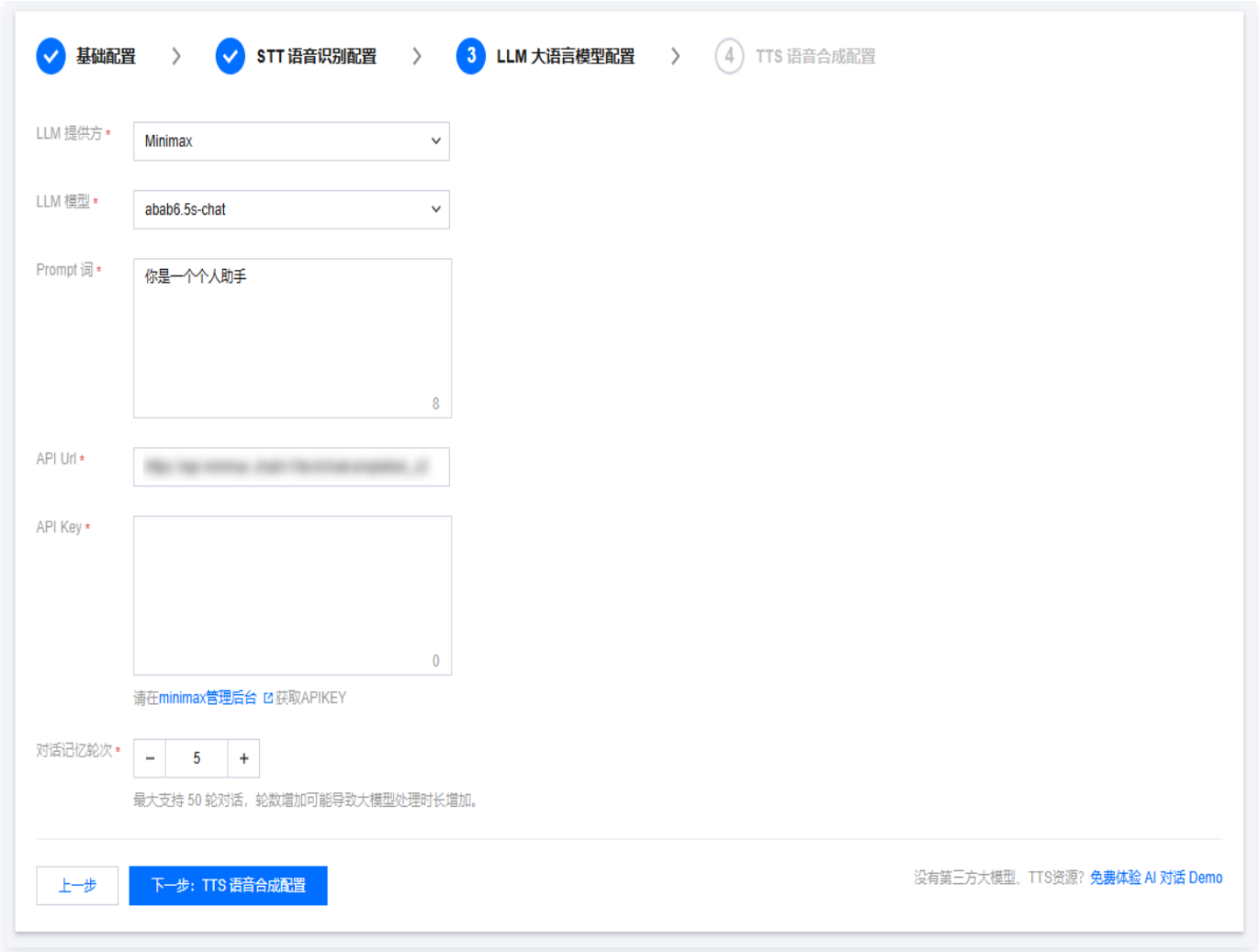
  调用 指引文档 。
  LLM 模型名称填写：deepseek-v3 或 deepseek-r1（由于模型有思考过程，对话的实时性会有影响）。
  API Url 填写：`https://api.lkeap.cloud.tencent.com/v1/chat/completions`
  API Key 参考文档获取 。
- **硅基流动平台**

  调用 指引文档 。
  LLM 模型名称 参考文档 。
  API Url 填写：`https://api.siliconflow.cn/v1/chat/completions`
  API Key 获取地址 。



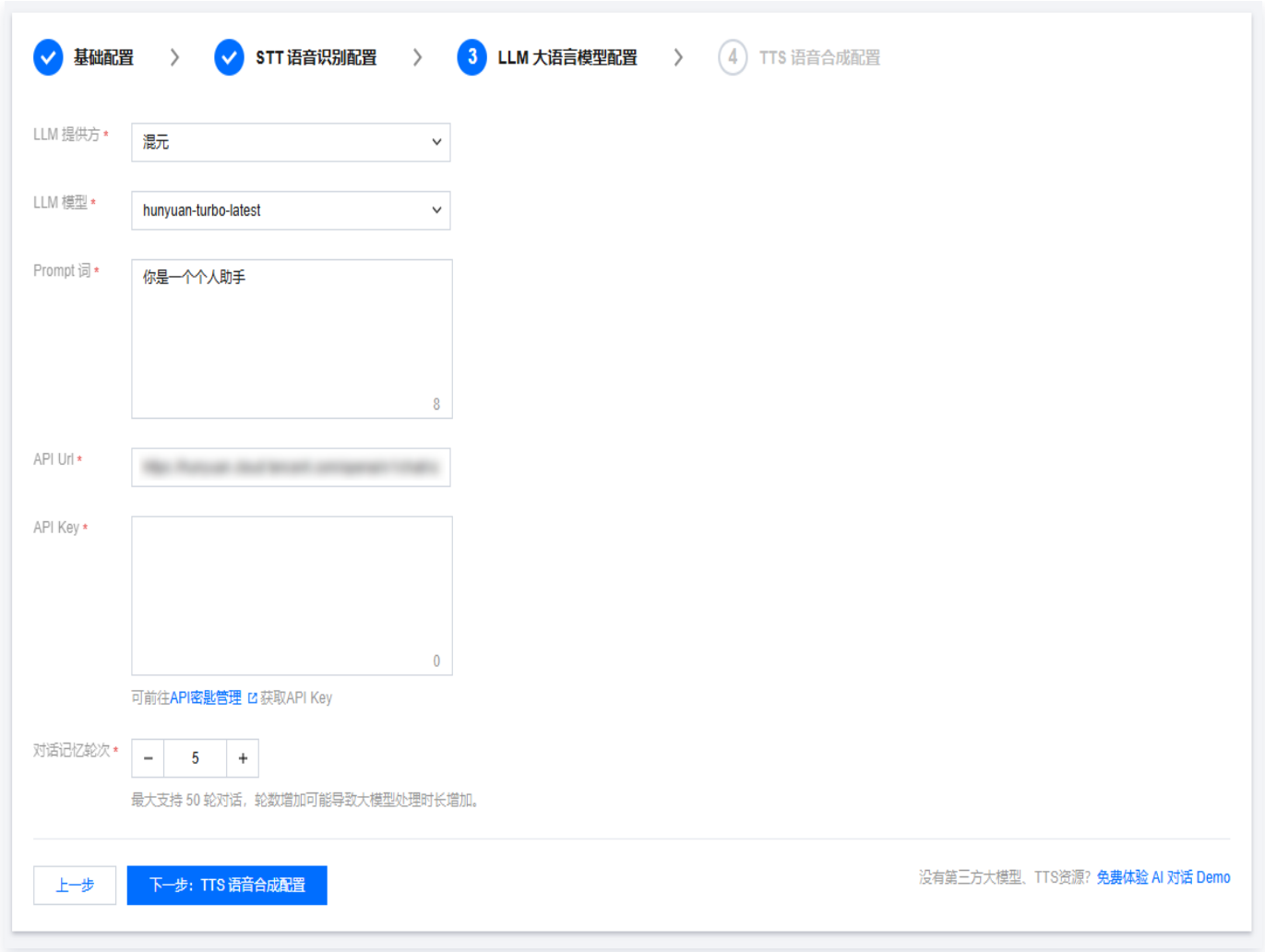
Minimax

LLM 提供方选择 Minimax 时，其他配置我们提供了默认参数，您根据需求进行微调；LLM 模型可下拉选项自由选择，API Key 可前往 minimax 管理后台 获取。



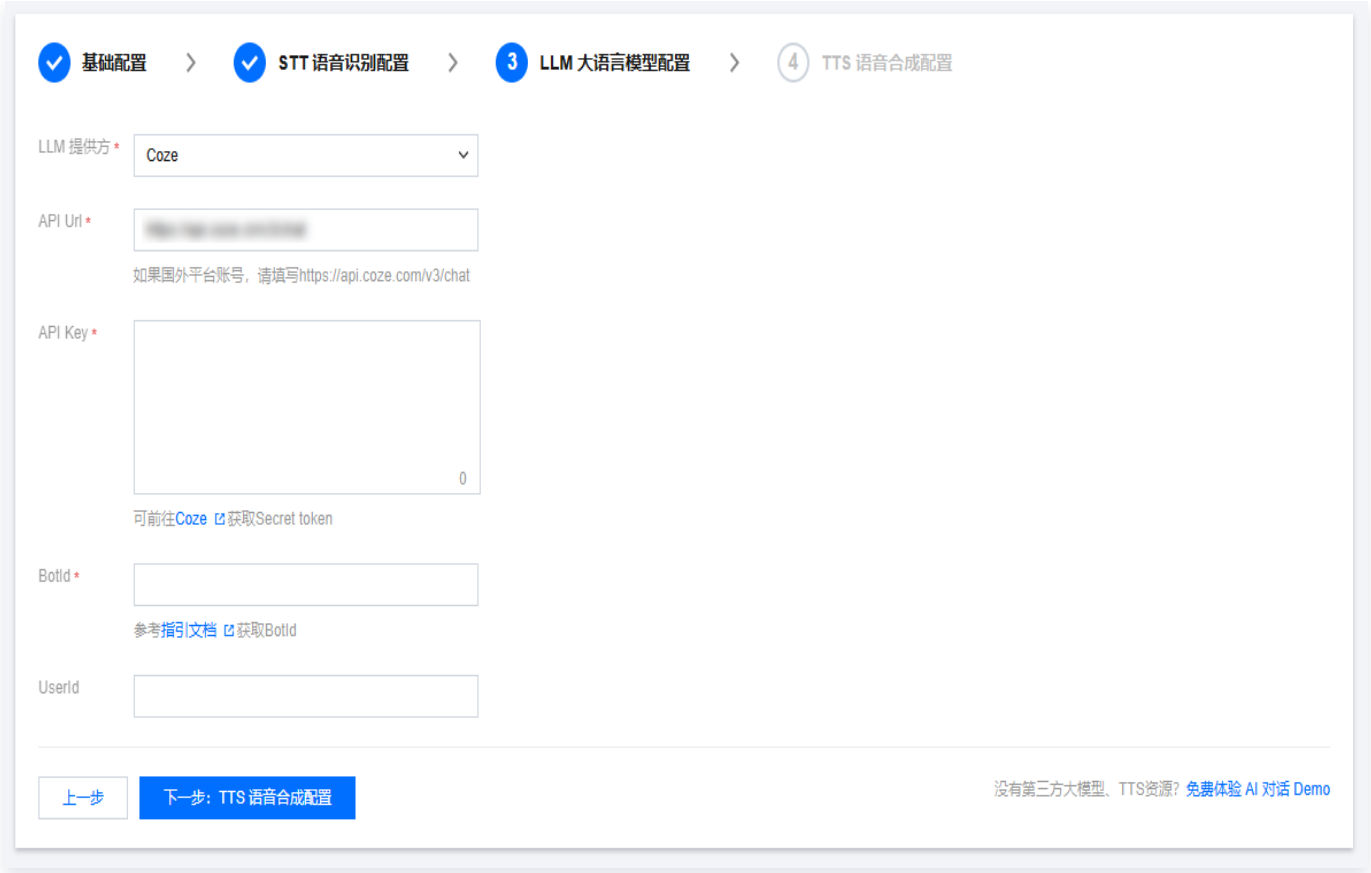
混元

LLM 提供方选择混元时，其他配置我们提供了默认参数，您根据需求进行微调；LLM 模型可下拉选项自由选择，API Key 可前往 API 密匙管理 获取。



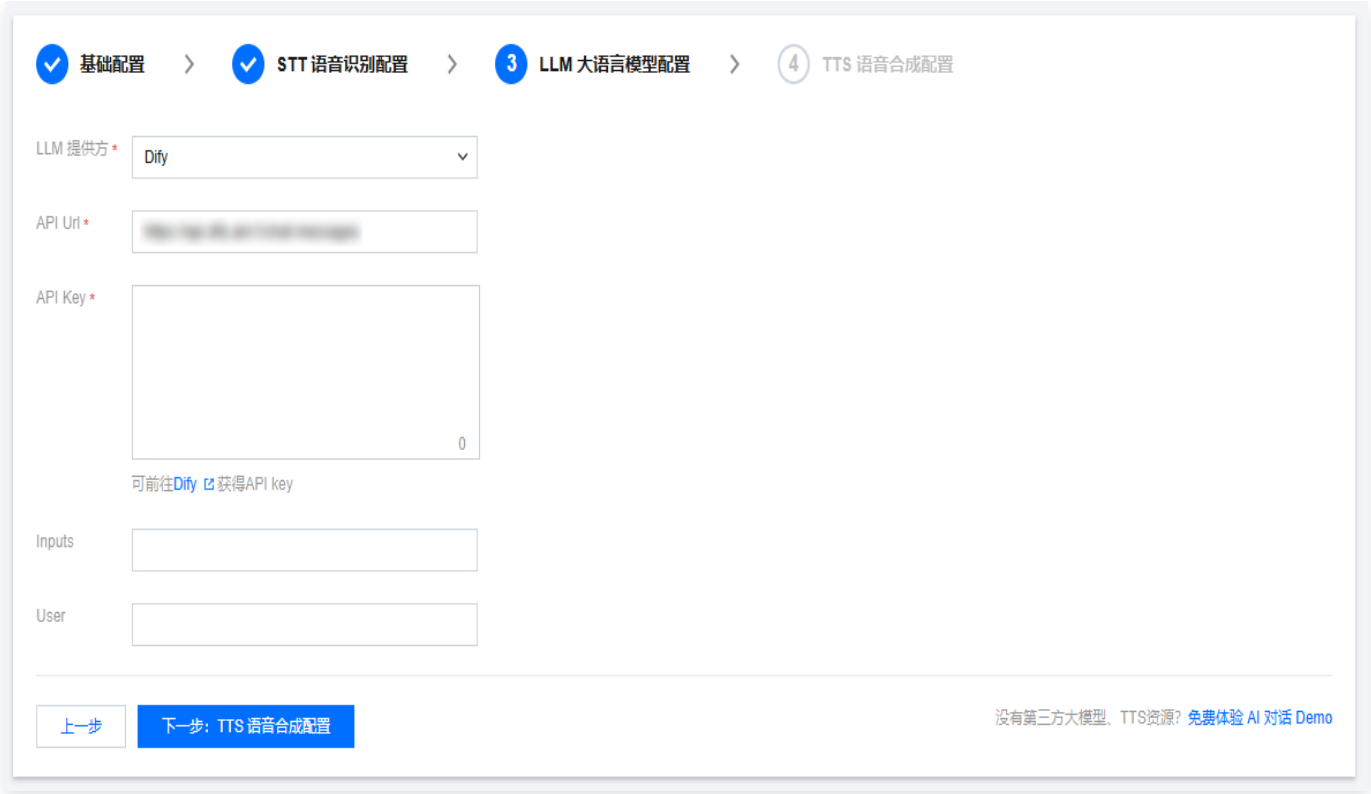
Coze

LLM 提供方选择 Coze 时，其他配置我们提供了默认参数，您根据需求进行微调；API Key 可前往 Coze 获取 Secret token，参见 指引文档 获取 BotId。



Dify

LLM 提供方选择 Dify 时，其他配置我们提供了默认参数，您根据需求进行微调；API Key 可前往 Dify 获取。



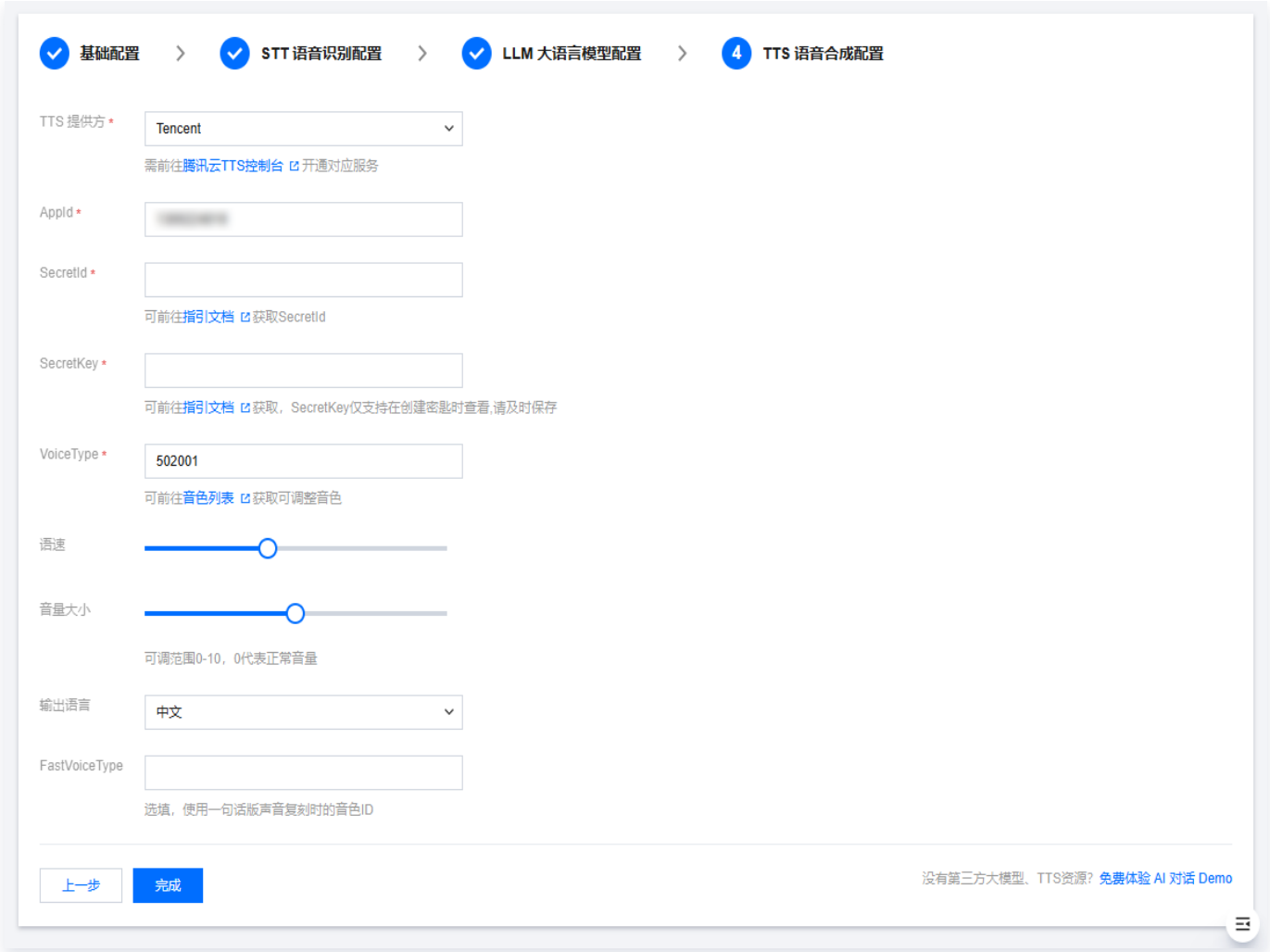
配置完成后单击**下一步**。

STEP 4：TTS 语音合成配置

在 TTS 语音合成配置页面：TTS 提供方支持 Tencent、Minimax、Azure、Cartesia、Elevenlabs 和 自定义。

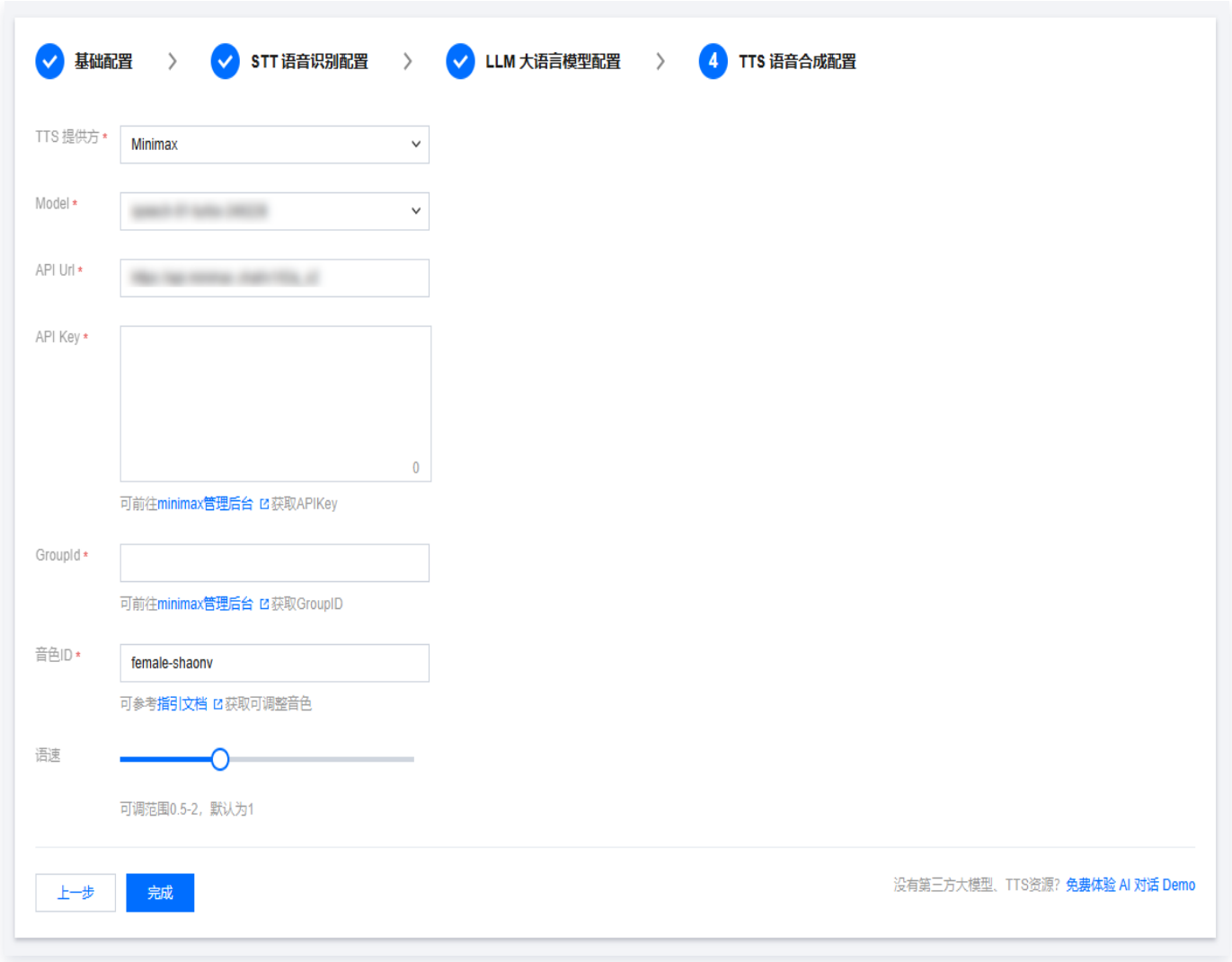
Tencent

- TTS 提供方选择 Tencent 时，您需 开通应用的 TTS 服务 来使用 TTS 语音合成功能。您可以点击 免费领取语言合成资源包 。
- SecretId 可前往 指引文档 获取。
- SecretKey 可前往 指引文档 获取，SecretKey 仅支持在创建密匙时查看,请及时保存。
- 可前往 音色列表 获取可调整音色。
- 其他配置我们提供了默认参数，您根据需求进行微调。



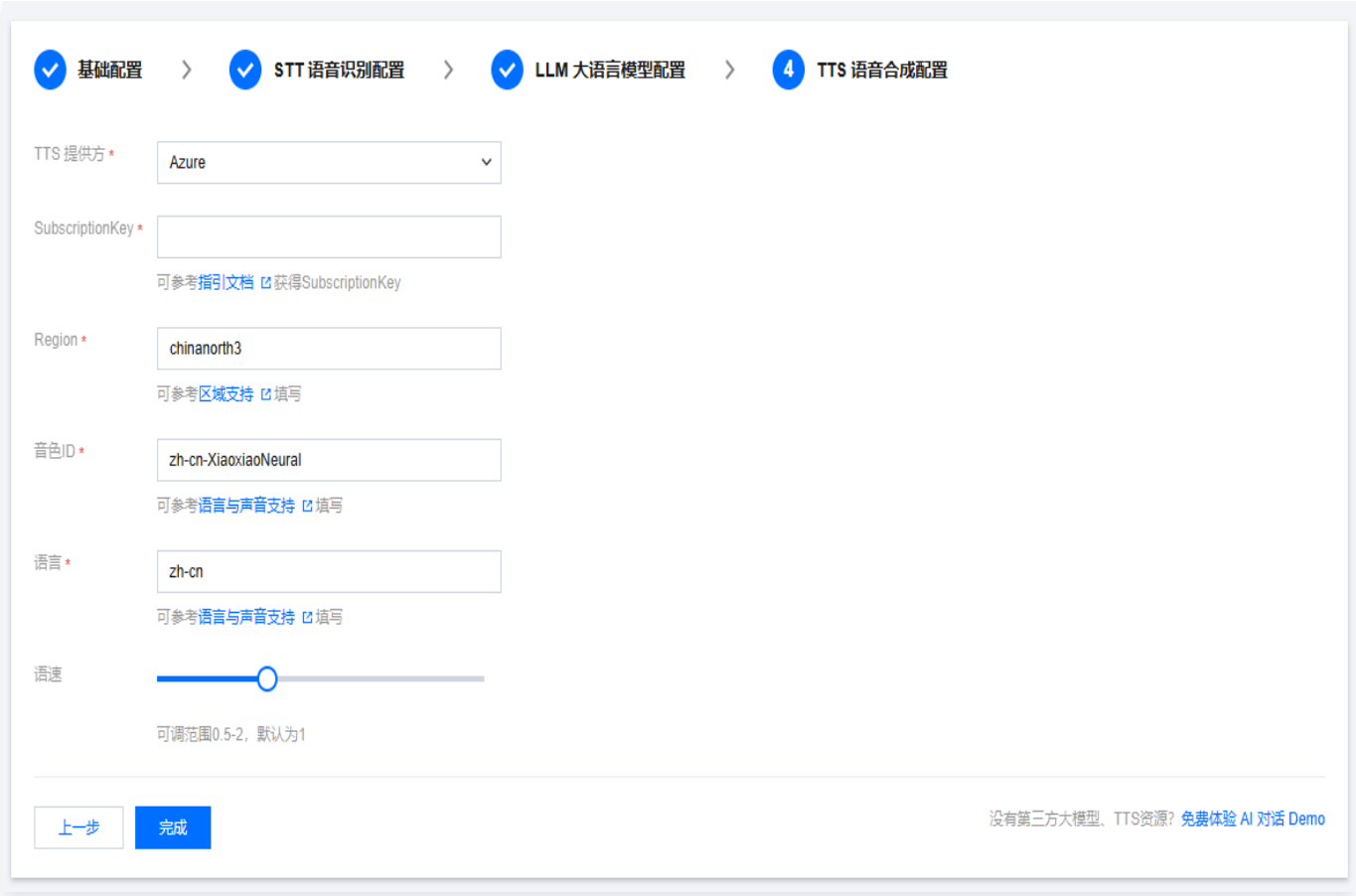
Minimax

- APIKey 可前往 minimax管理后台 获取。
- GroupID 可前往 minimax管理后台 获取。
- 可参考 指引文档 获取可调整音色。
- 其他配置我们提供了默认参数，您根据需求进行微调。



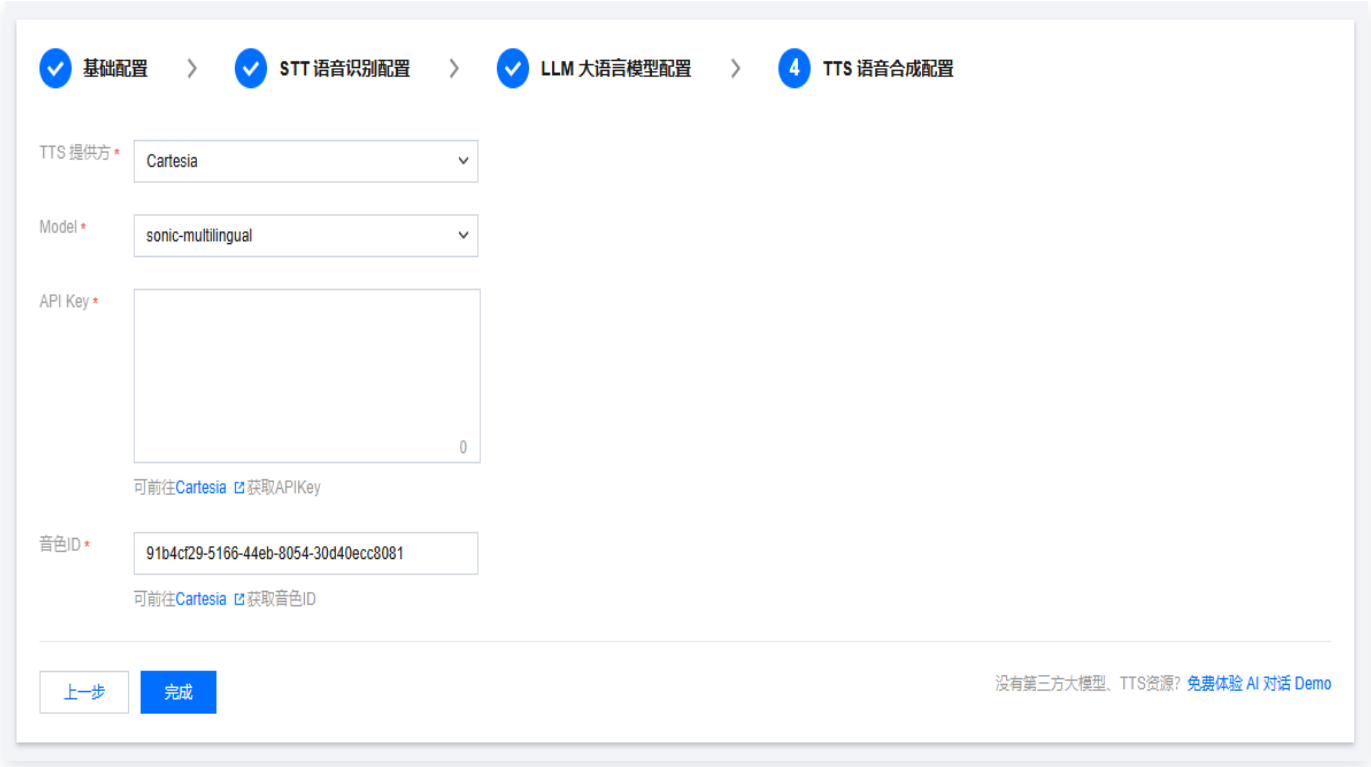
Azure

- SubscriptionKey 可参考 指引文档 获得。
- Region 可参考 区域支持 填写。
- 语言可参考 语言与声音支持 填写。
- 其他配置我们提供了默认参数，您根据需求进行微调。



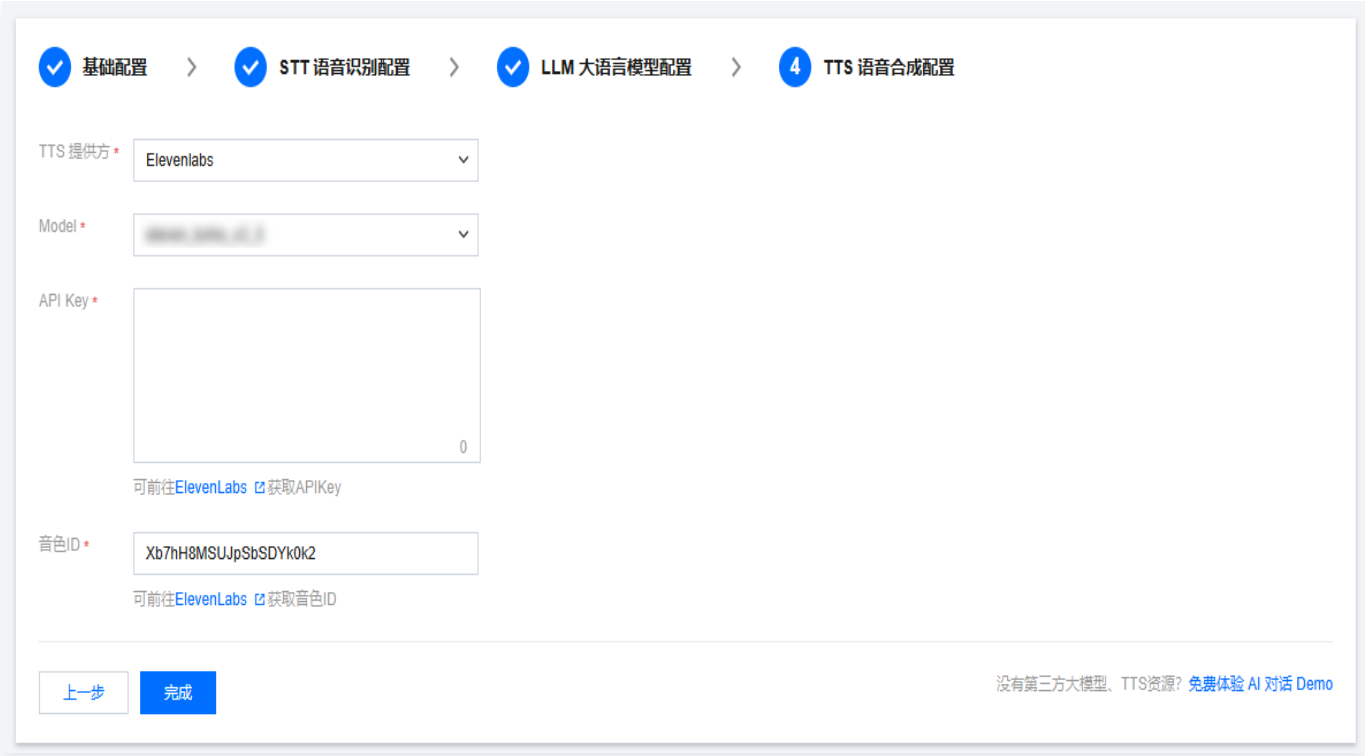
Cartesia

- APIKey 可前往 Cartesia 获取APIKey。
- 音色 ID 可前往 Cartesia 获取。
- 其他配置我们提供了默认参数，您根据需求进行微调。



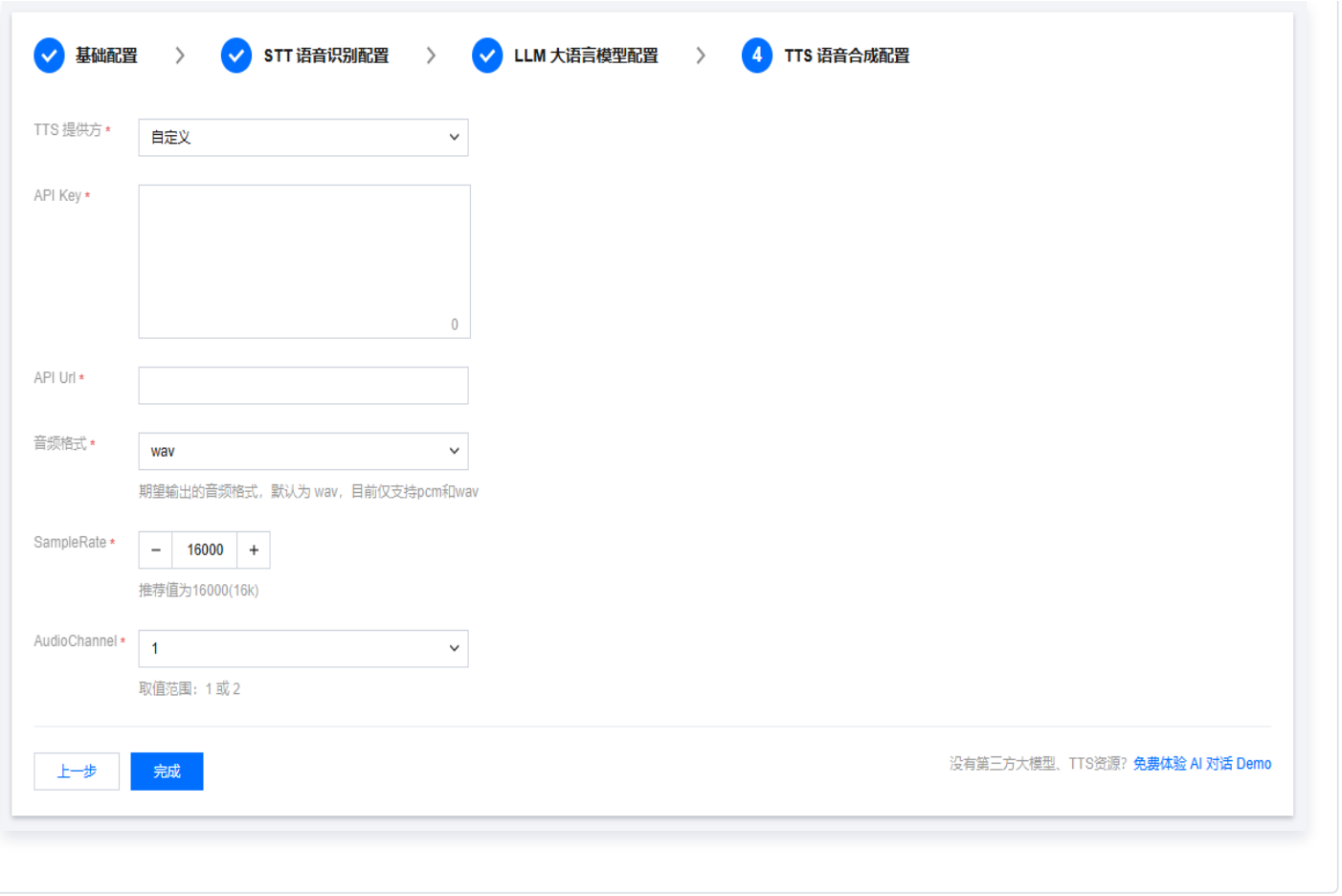
Elevenlabs

- APIKey 可前往 ElevenLabs 获取。
- 音色 ID 可前往 ElevenLabs 获取。
- 其他配置我们提供了默认参数，您根据需求进行微调。



自定义

- 可参见 自定义 TTS 协议 配置。
- TTS 提供方选择自定义时，您可根据实际情况填写 API Key 和 API Url。
- 其他配置我们提供了默认参数，您根据需求进行微调。

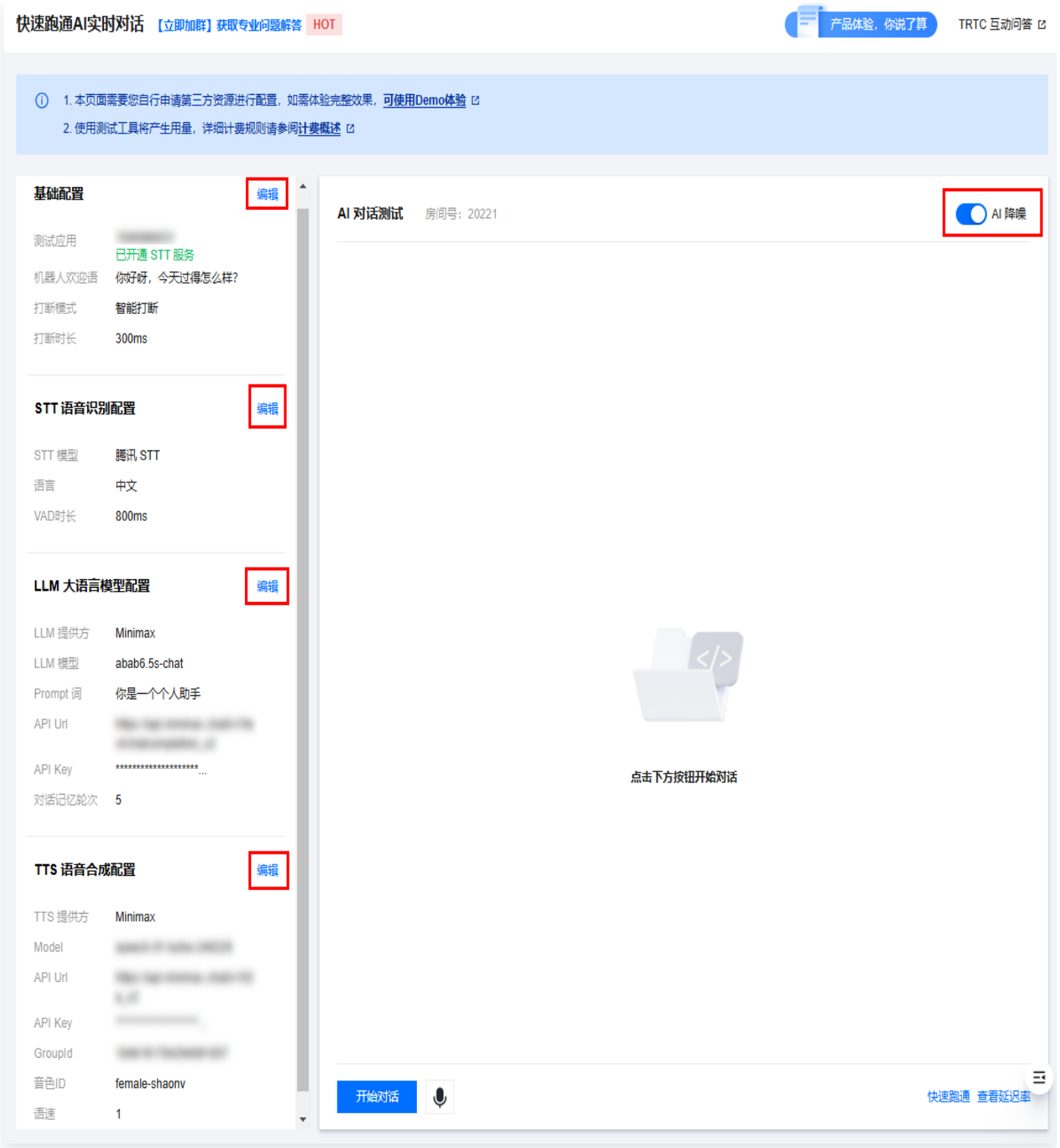


配置完成后单击**完成**。

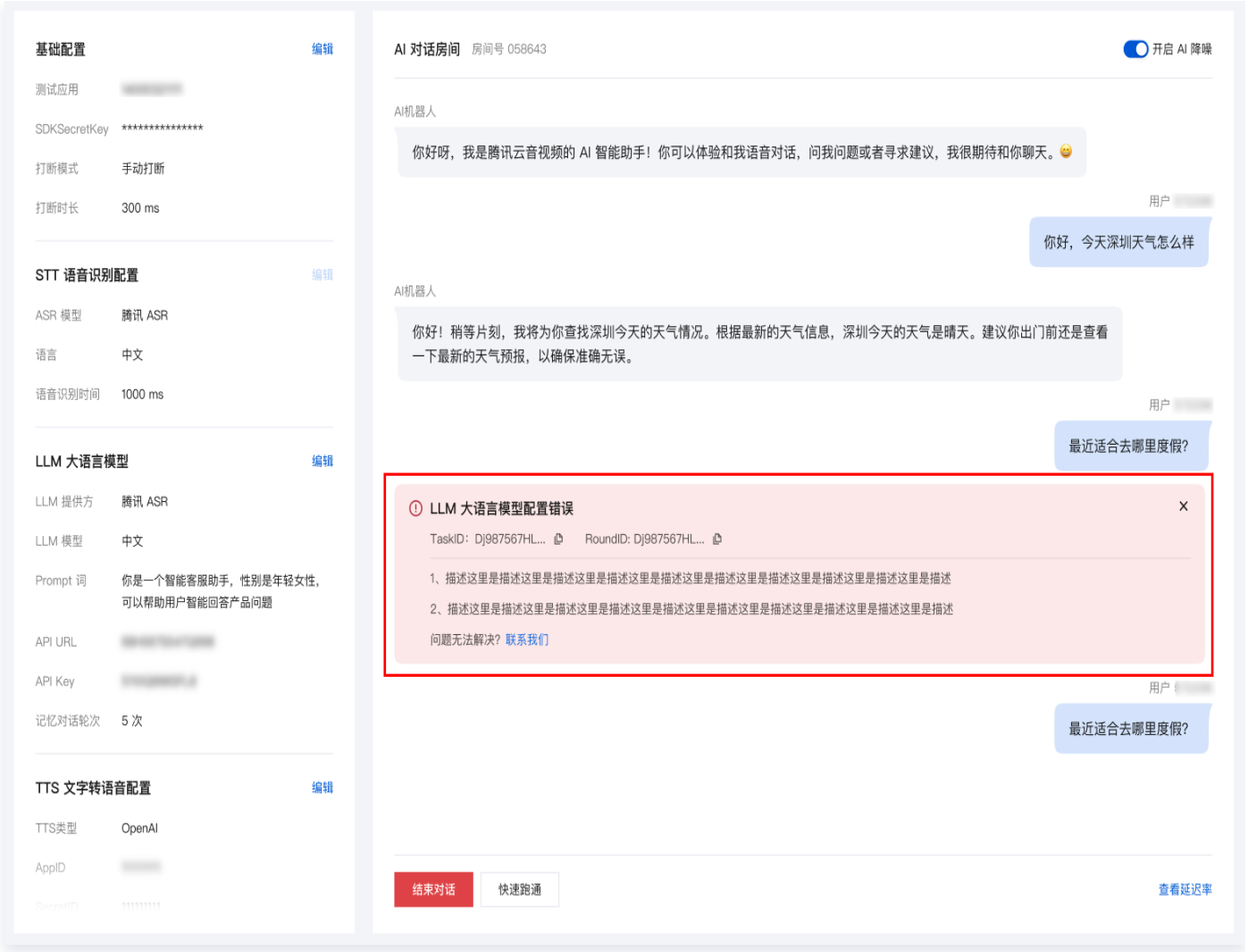
主页面介绍

开始对话

- AI 降噪：使用此功能需开通 实时音视频包月套餐包 尊享版或旗舰版。
- 对话过程中支持编辑修改打断时长、LLM 大语言模型配置和 TTS 语音合成配置，便于进行效果测评。

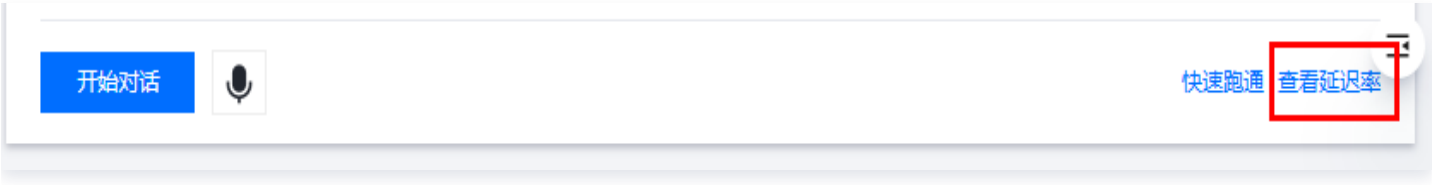


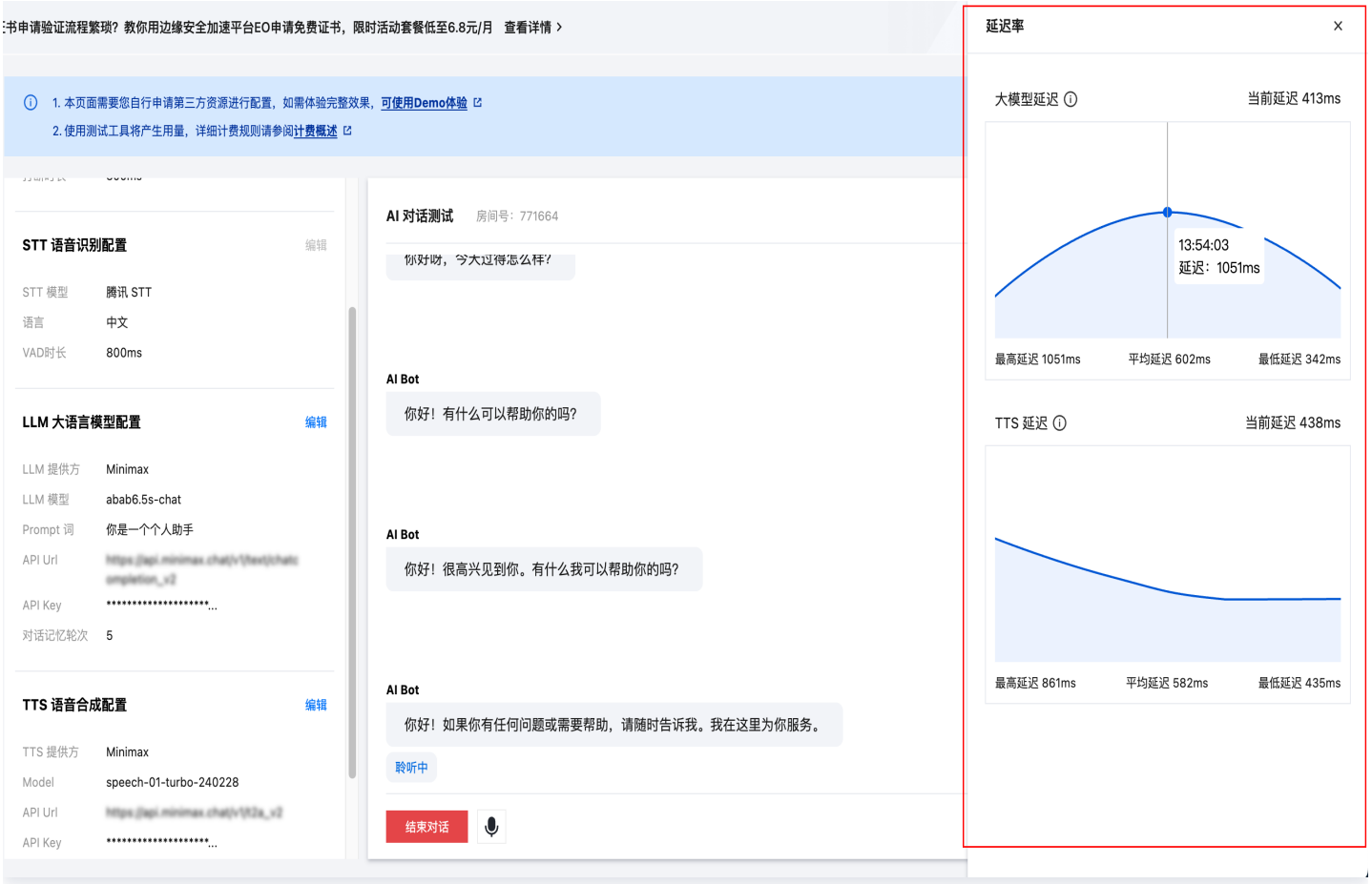
- **对话过程中若出现错误码的展示：请注意，错误码有些是阻塞对话进行的，请根据错误提示及时修改配置，如无法解决，请复制 Taskid 和 Roundid 信息，联系我们 进行查询。有一些错误码不影响对话正常进行，例如反应超时，您可以根据实际情况处理。**



查看延迟率

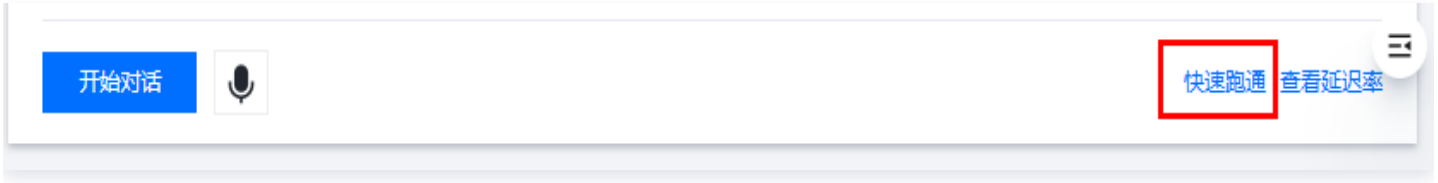
您在对话时可随时查看延迟率，便于您选择低延迟模型。

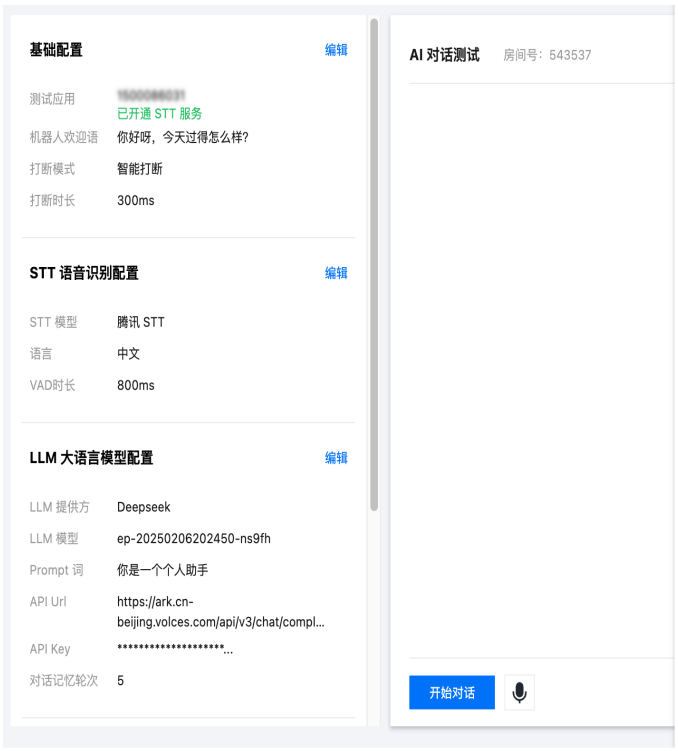




快速跑通

我们支持用户在自己的本地环境快速跑通 AI 实时对话。在系统配置填写的参数已经预置，无需再次填写（目前支持 Web 端）。





2 运行nodejs代码

```
/**
 * 腾讯云 TRTC 接口封装
 * 请先下载依赖：npm i express tencentcloud-sdk-nodejs-trtc
 * 启动：node index.js
*/
const express = require('express');
const tencentcloud = require("tencentcloud-sdk-nodejs-trtc");

const TrtcClient = tencentcloud.trtc.v20190722.Client;

const clientConfig = {
  credential: {
    secretId: "填写你的secretId",
    secretKey: "填写你的secretKey",
  },
  region: "ap-beijing",
  profile: {
    httpProfile: {
      endpoint: "trtc.tencentcloudapi.com",
    },
```

3 前端代码

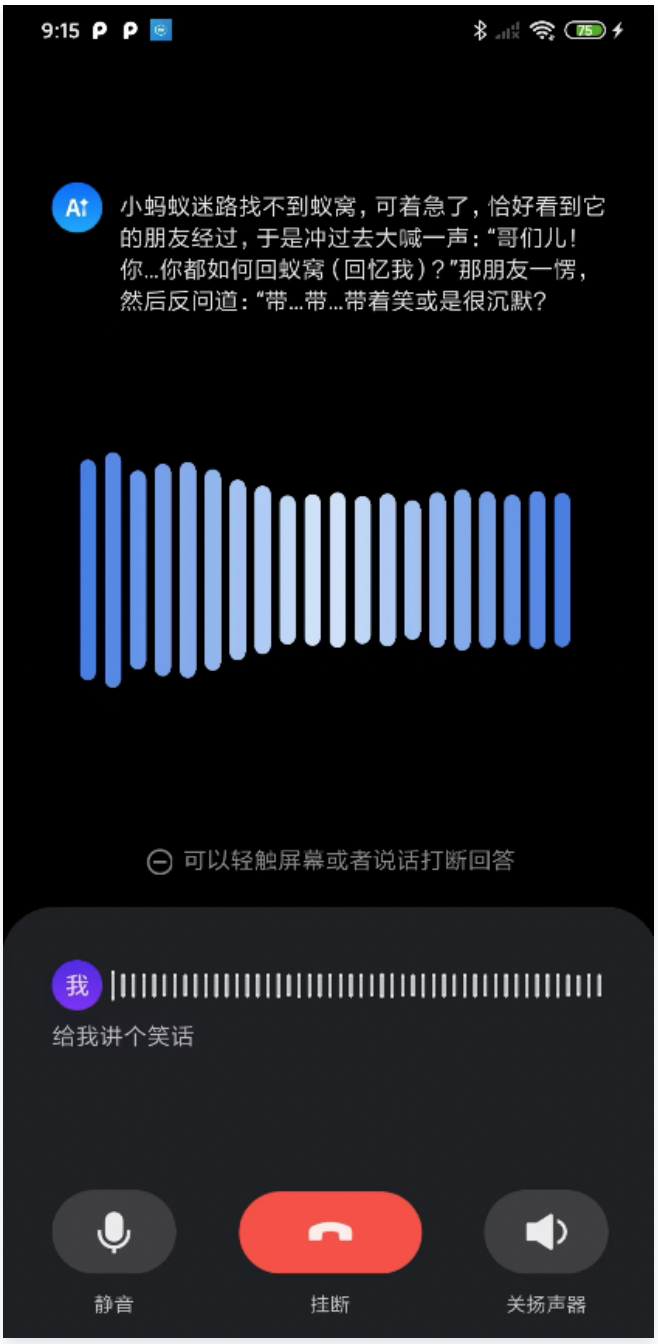
```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>AIConversation</title>
```

跑通 Demo
Android

最近更新时间：2025-02-25 18:15:22

本文档主要介绍如何快速跑通 AIConversationKit 示例工程，体验高质量的 AI 对话。跟随本文档，您可以在 20 分钟内跑通 Demo，并最终体验一个包含完备 UI 界面的 AI 对话。

AI 对话界面



环境准备

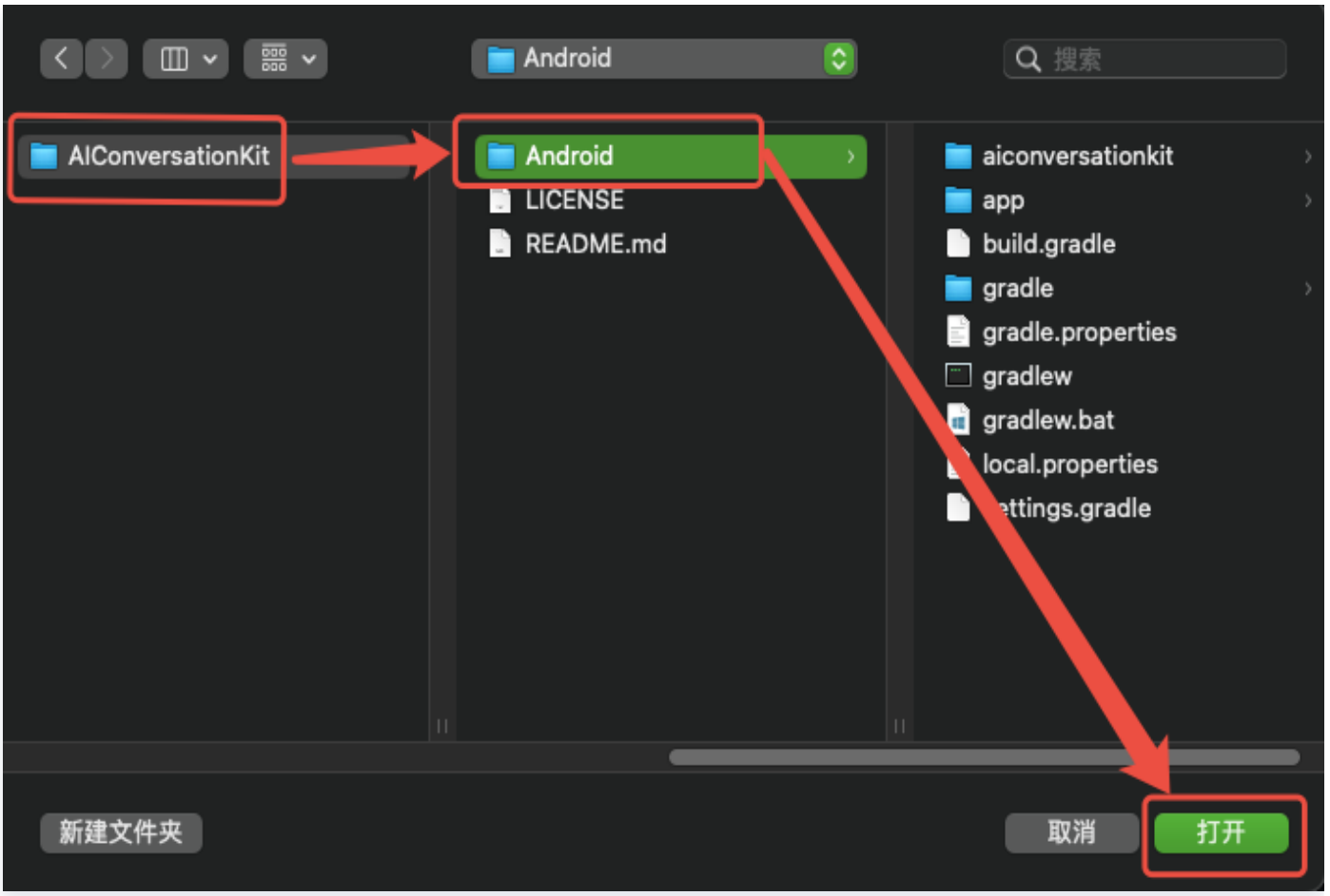
- 最低兼容 Android 4.4（SDK API Level 19），建议使用 Android 5.0（SDK API Level 21）及以上版本。
- Android Studio 3.5及以上版本。

下载 Demo

1. 从 github 下载 AIConversationKit Demo 源码，或者直接在命令行运行以下命令：

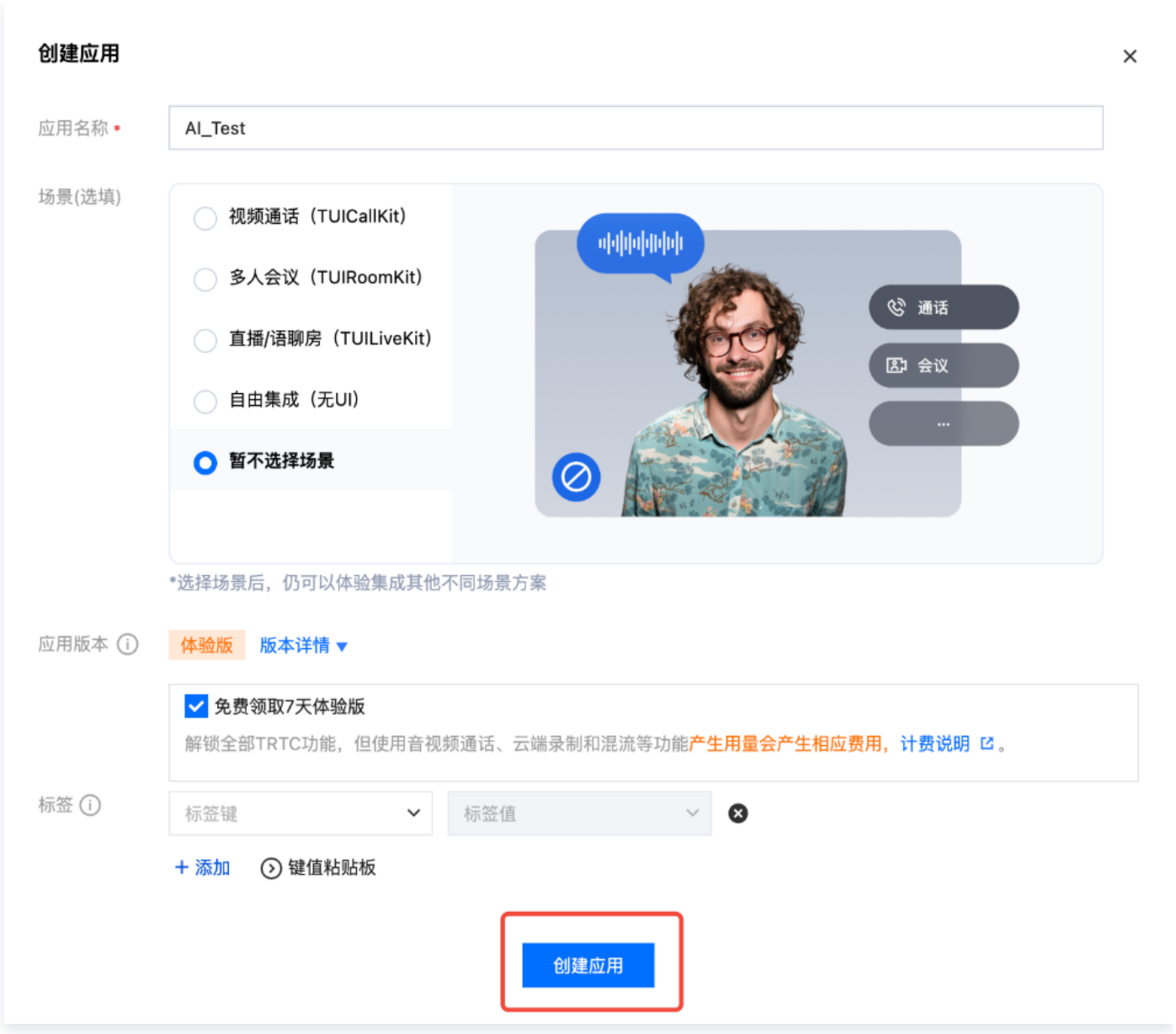
```
git clone https://github.com/Tencent-RTC/AIConversationKit.git
```

2. 通过 Android Studio 打开 **AIConversationKit Android** 项目：

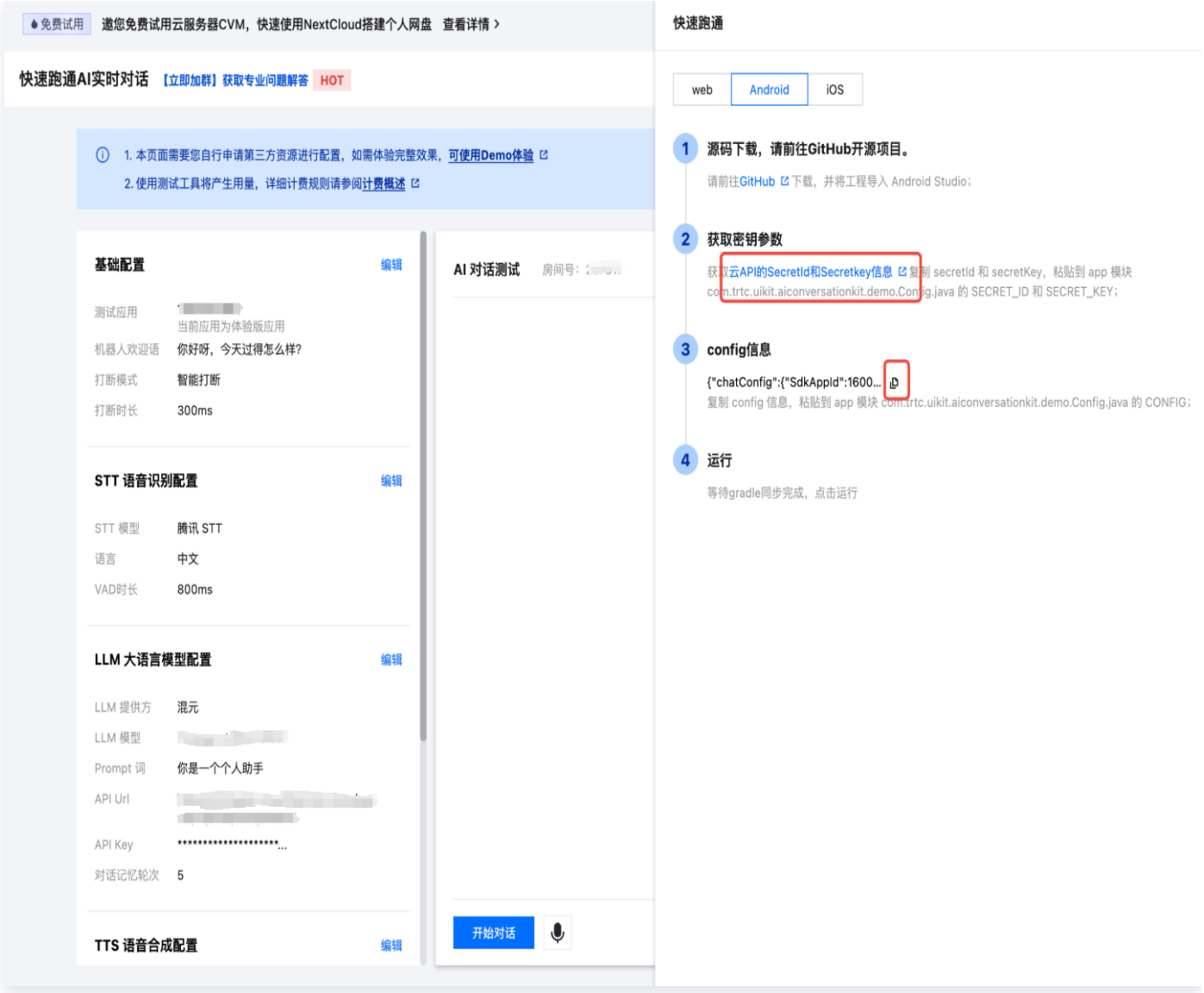


配置 Demo

1. 请先到 应用管理 创建应用，领取 7 天免费试用版本。



2. 使用 Playground 配置 AI 对话参数，点击右下角的**快速跑通**，切换到 Android，获取 SecretId、SecretKey 和 Config 参数。

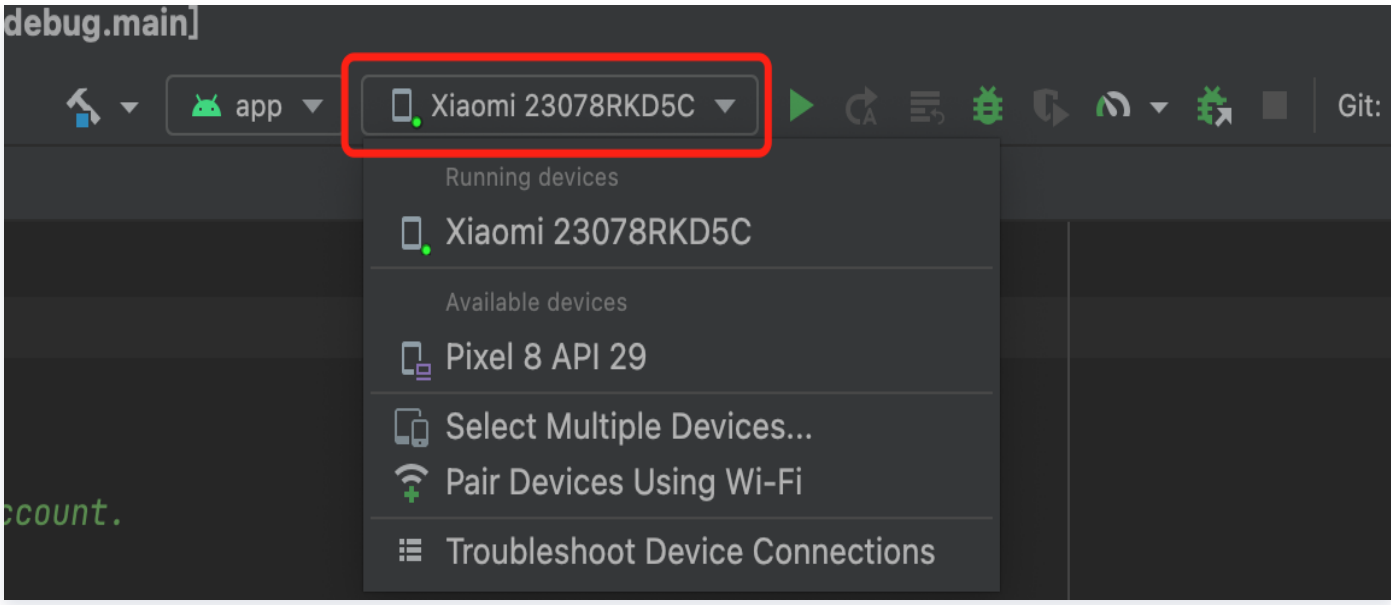


3. 打开工程，在工程内找到 `Android/app/src/main/java/com/trtc/uikit/aiconversationkit/demo/Config.java` 文件。将上一步中获取到的对应的 SecretId、SecretKey 和 Config 填入其中：

```
public class Config {
    public static final String SECRET_ID  = "";
    public static final String SECRET_KEY = "";
    public static final String CONFIG     = "";
}
```

跑通 Demo

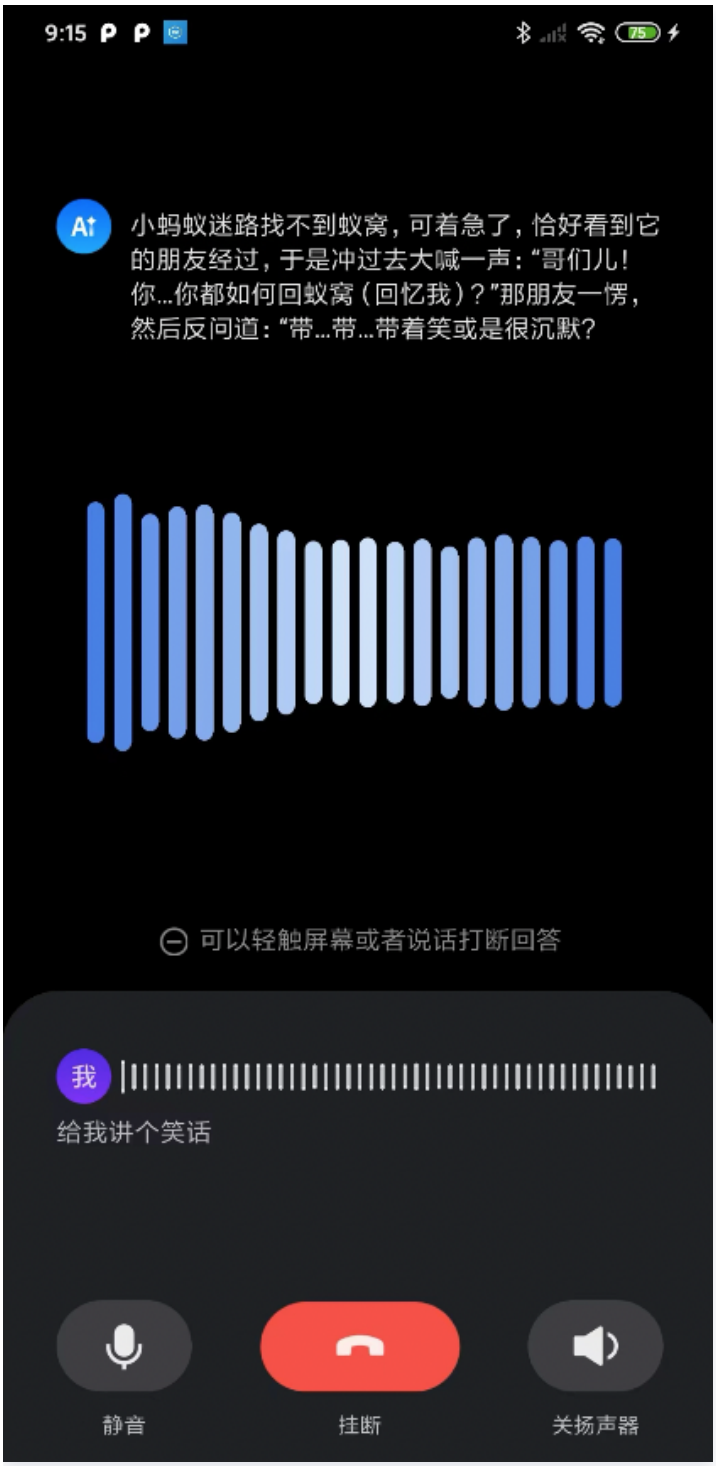
1. 在 Android Studio 右上角如下图所示处选择您要将 Demo 运行的设备：



2. 选择完成后单击运行，将 AIConversationKit Android Demo 运行到目标设备上。

开始您的第一次对话

开始和 AI 说话，比如让 AI 讲个笑话。

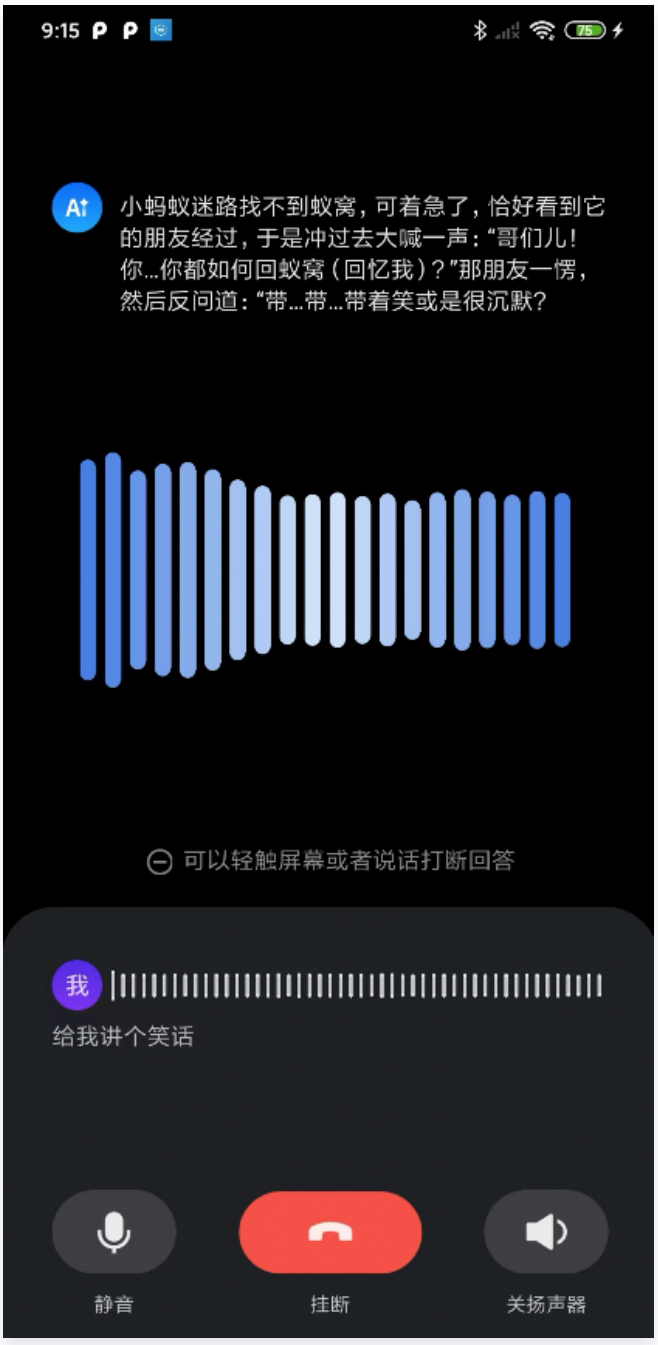

9:15
小蚂蚁迷路找不到蚁窝，可着急了，恰好看到它的朋友经过，于是冲过去大喊一声："哥们儿！你...你都如何回蚁窝（回忆我）？"那朋友一愣，然后反问道："带...带...带着笑或是很沉默？
可以轻触屏幕或者说话打断回答
我
给我讲个笑话
静音
挂断
关扬声器

iOS

最近更新时间：2025-02-25 18:15:22

本文档主要介绍如何快速跑通 AIConversationKit 示例工程，体验高质量的 AI 对话。跟随本文档，您可以在 20 分钟内跑通 Demo，并最终体验一个包含完备 UI 界面的 AI 对话。

AI 对话界面



环境准备

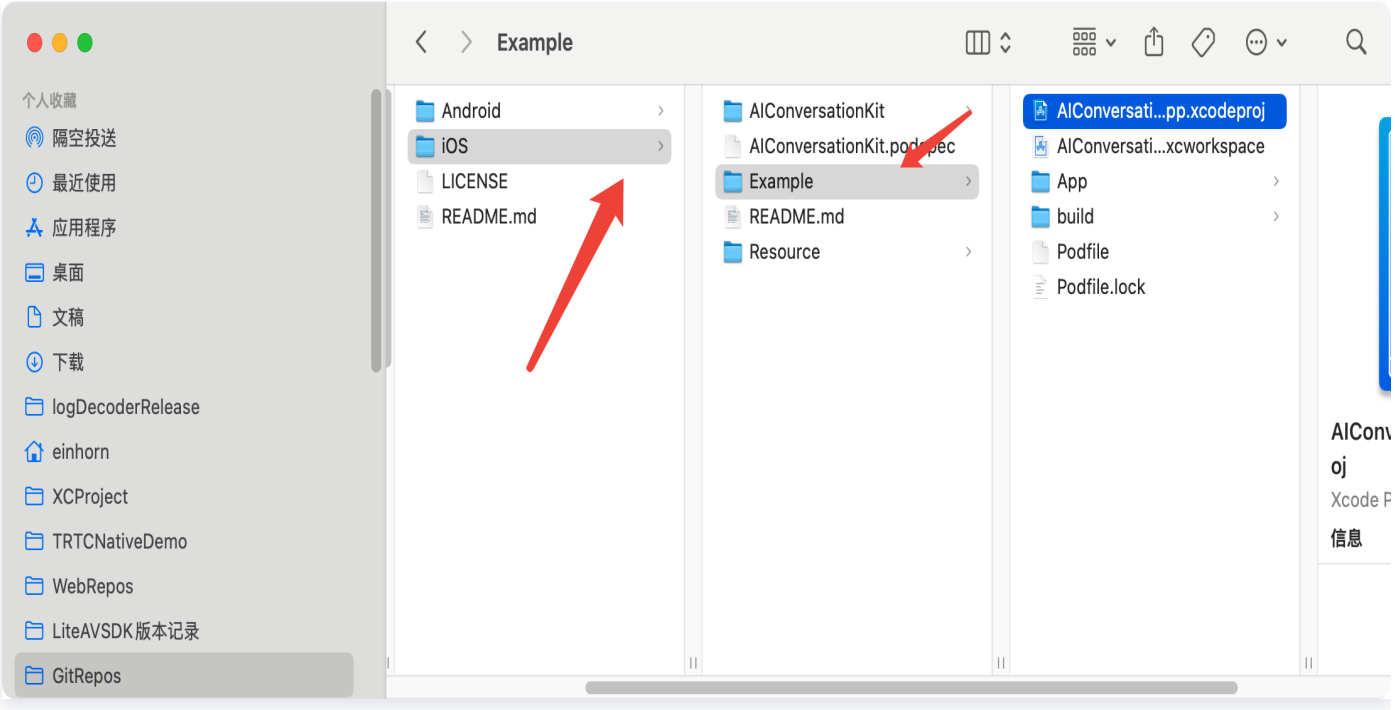
- 最低兼容 iOS 13，建议使用 iOS13 及以上版本。
- Xcode 13及以上版本。

下载 Demo

1. 从 github 下载 AIConversationKit Demo 源码，或者直接在命令行运行以下命令：

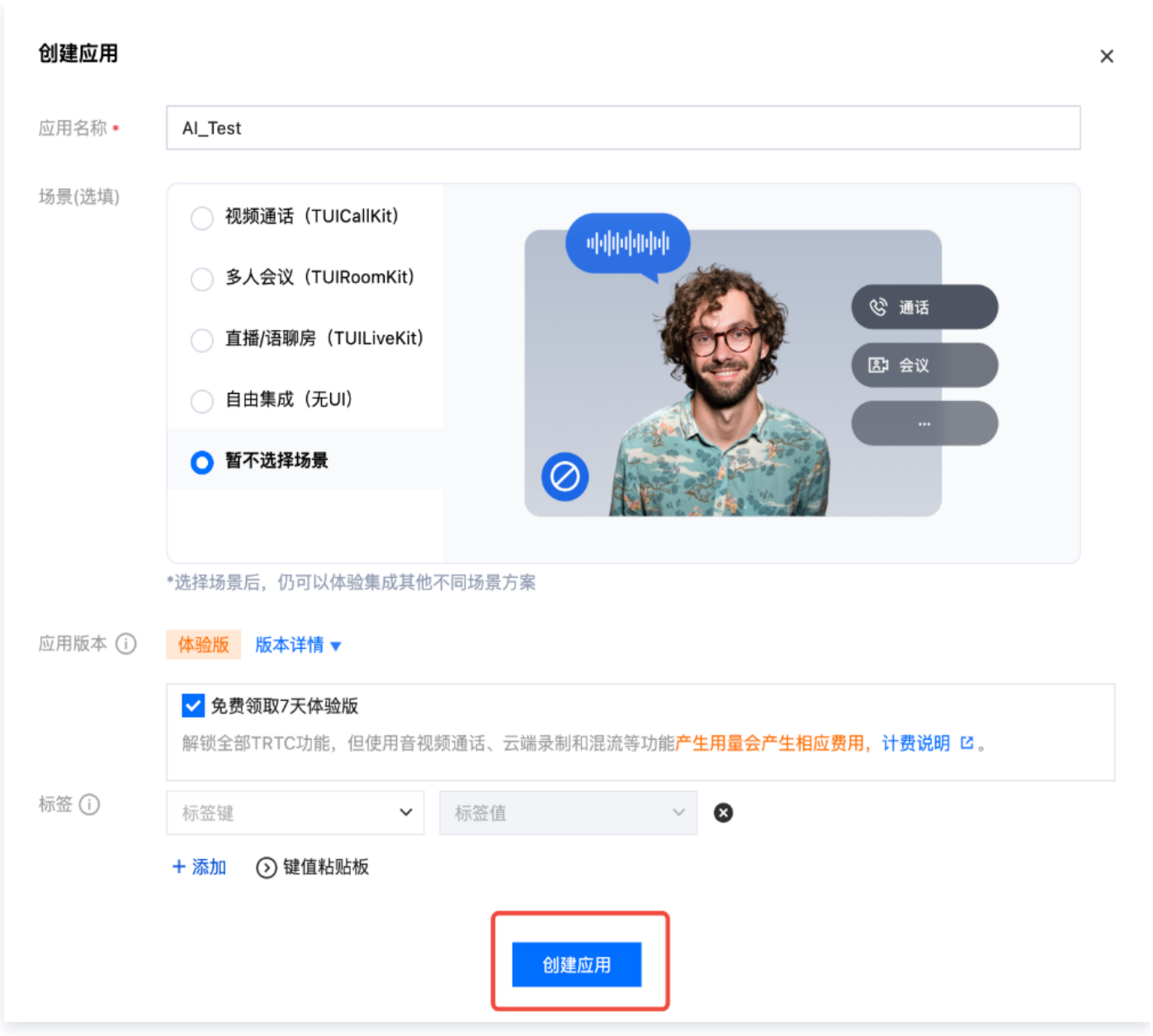
```
git clone https://github.com/Tencent-RTC/AIConversationKit.git
```

2. 解压文件夹后，使用 mac 的终端应用，cd 到 Example 下执行 pod install。



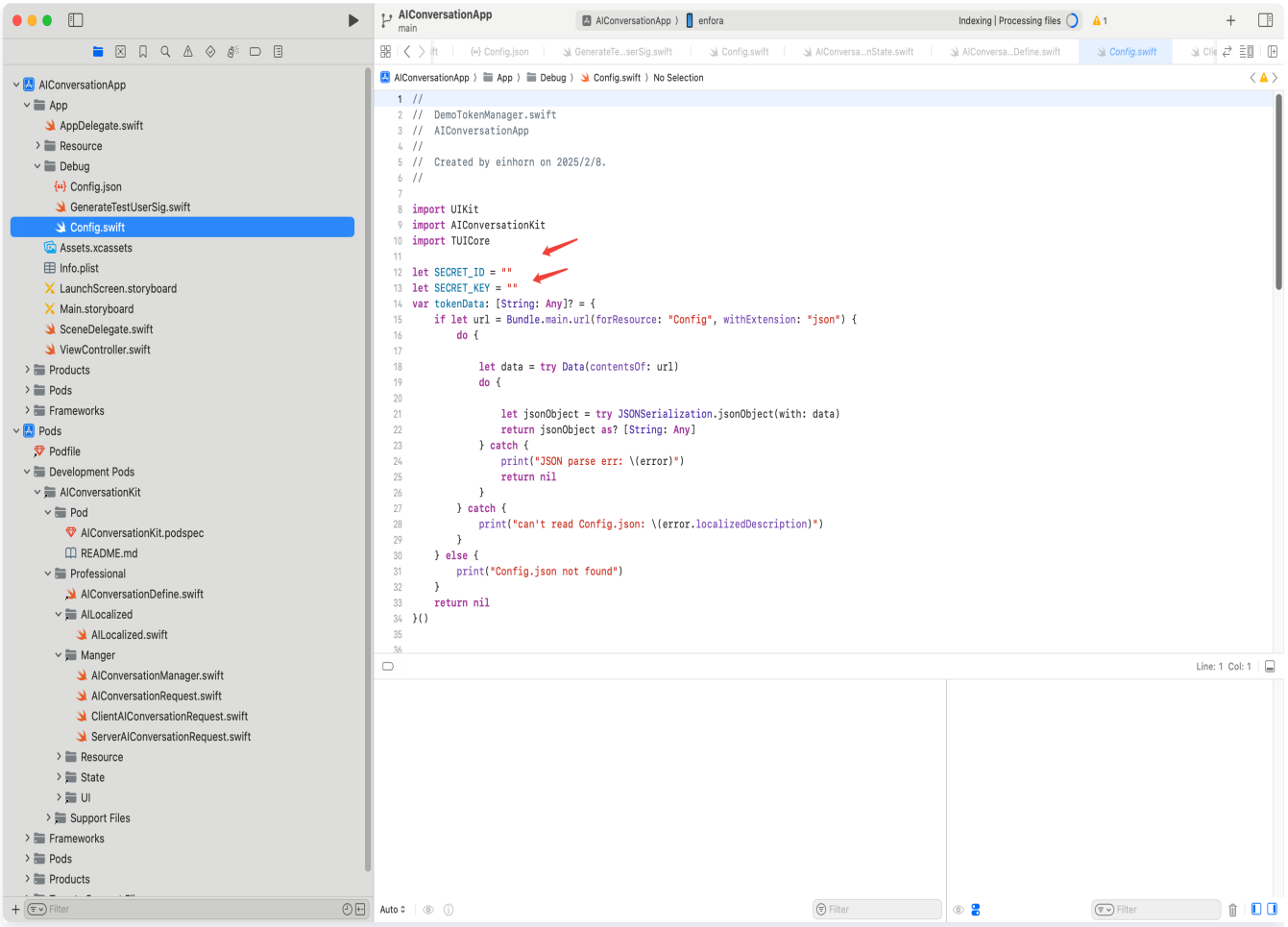
配置 Demo

1. 请先到 应用管理 创建应用，领取 7 天免费试用版本。

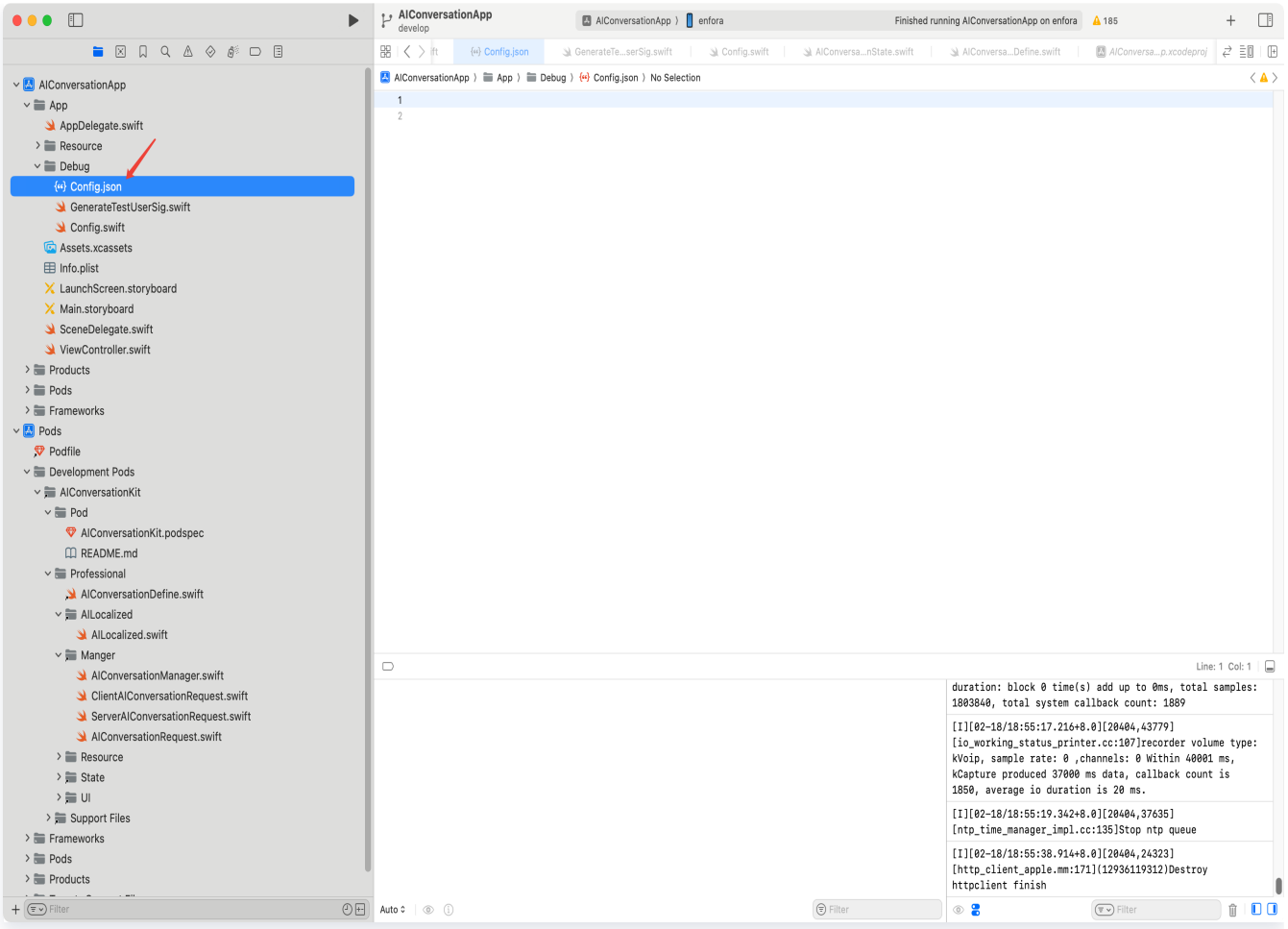


2. 使用 Playground 配置 AI 对话参数，点击右下角的**快速跑通**，切换到 iOS，获取 SecretId、SecretKey 和 Config 参数。

3. 使用 Xcode 打开 AIConversationApp.xcworkspace 工程，在工程内找到 App/Debug/Config.swift：
   - 先将上一步中获取到的对应的 **SecretId**、**SecretKey** 填入其中箭头处：

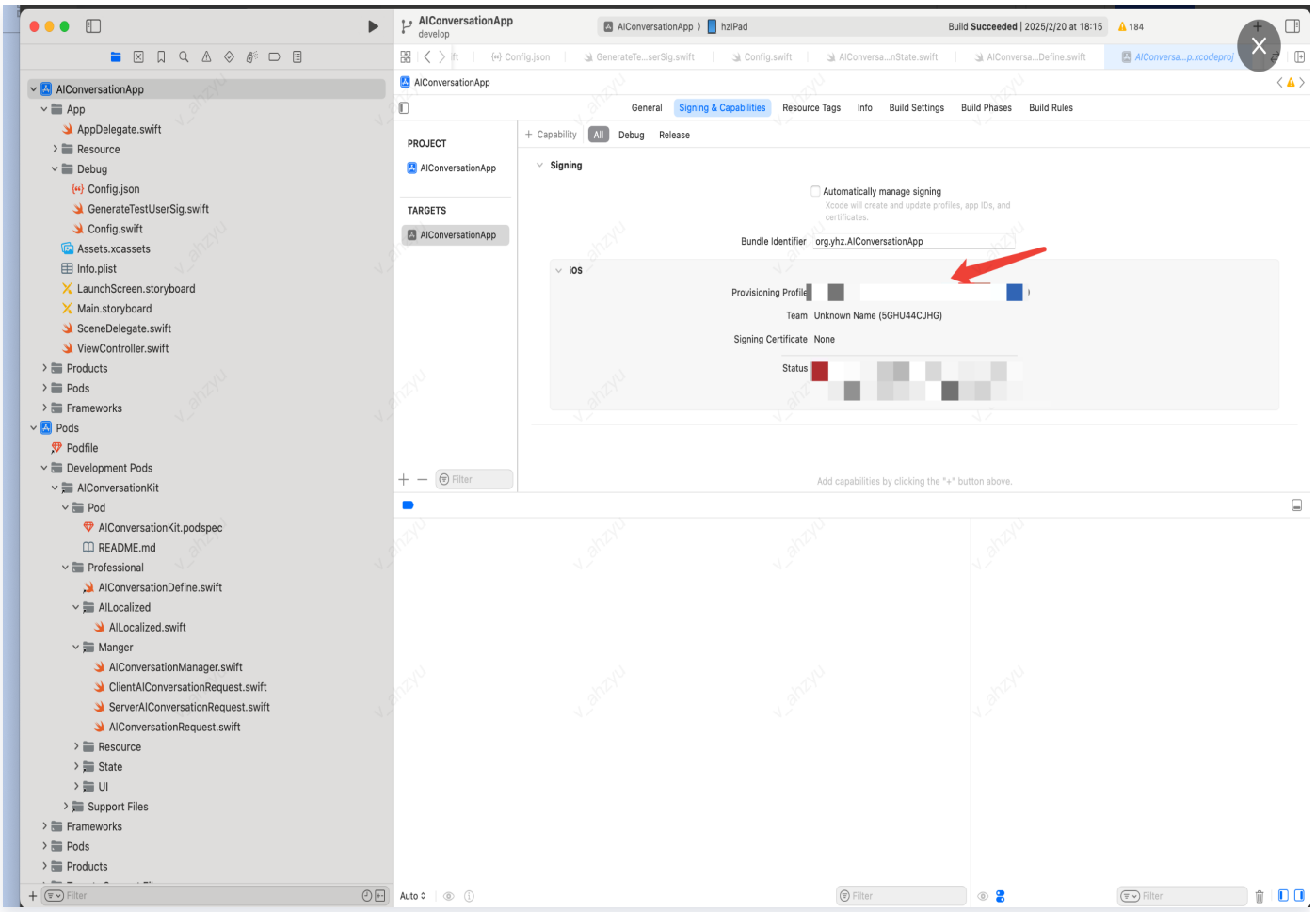


- 然后复制 config 信息到 App/Debug/Config.json。

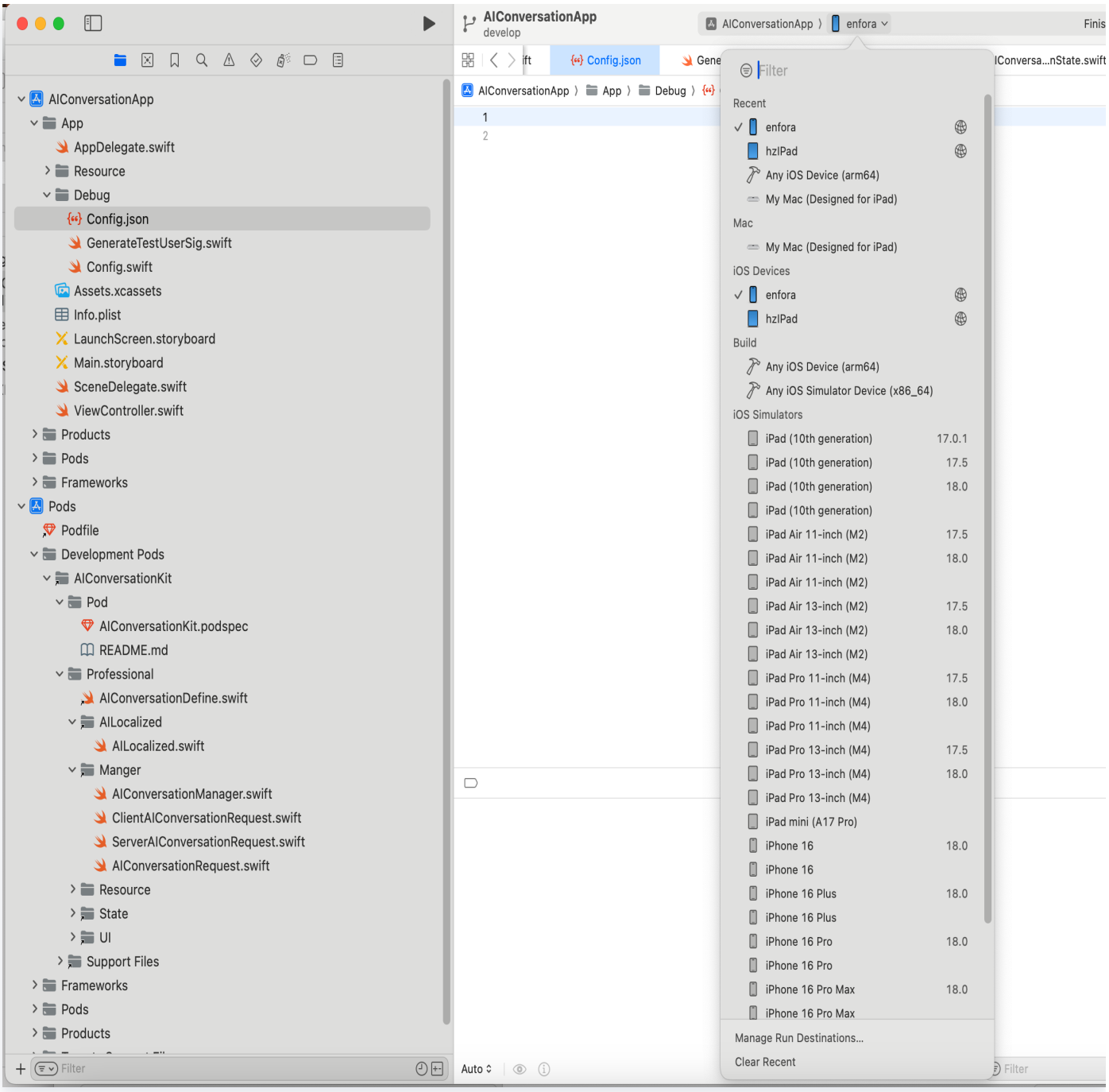


跑通 Demo

1. 配置证书：


AIConversationApp
App
AppDelegate.swift
Resource
Debug
Config.json
GenerateTestUserSig.swift
Config.swift
Assets.xcassets
Info.plist
LaunchScreen.storyboard
Main.storyboard
SceneDelegate.swift
ViewController.swift
Products
Pods
Frameworks
Pods
Podfile
Development Pods
AIConversationKit
Pod
AIConversationKit.podspec
README.md
Professional
AIConversationDefine.swift
AILocalized
AILocalized.swift
Manger
AIConversationManager.swift
ClientAIConversationRequest.swift
ServerAIConversationRequest.swift
AIConversationRequest.swift
Resource
State
UI
Support Files
Frameworks
Pods
Products
AIConversationApp
develop
AIConversationApp
hzlPad
Build Succeeded | 2025/2/20 at 18:15
184
General
Signing & Capabilities
Resource Tags
Info
Build Settings
Build Phases
Build Rules
PROJECT
AIConversationApp
TARGETS
AIConversationApp
Capability
All
Debug
Release
Signing
Automatically manage signing
Xcode will create and update profiles, app IDs, and certificates.
Bundle Identifier
org.yhz.AIConversationApp
iOS
Provisioning Profile
Team
Unknown Name (5GHU44CJHG)
Signing Certificate
None
Status
Add capabilities by clicking the "+" button above.

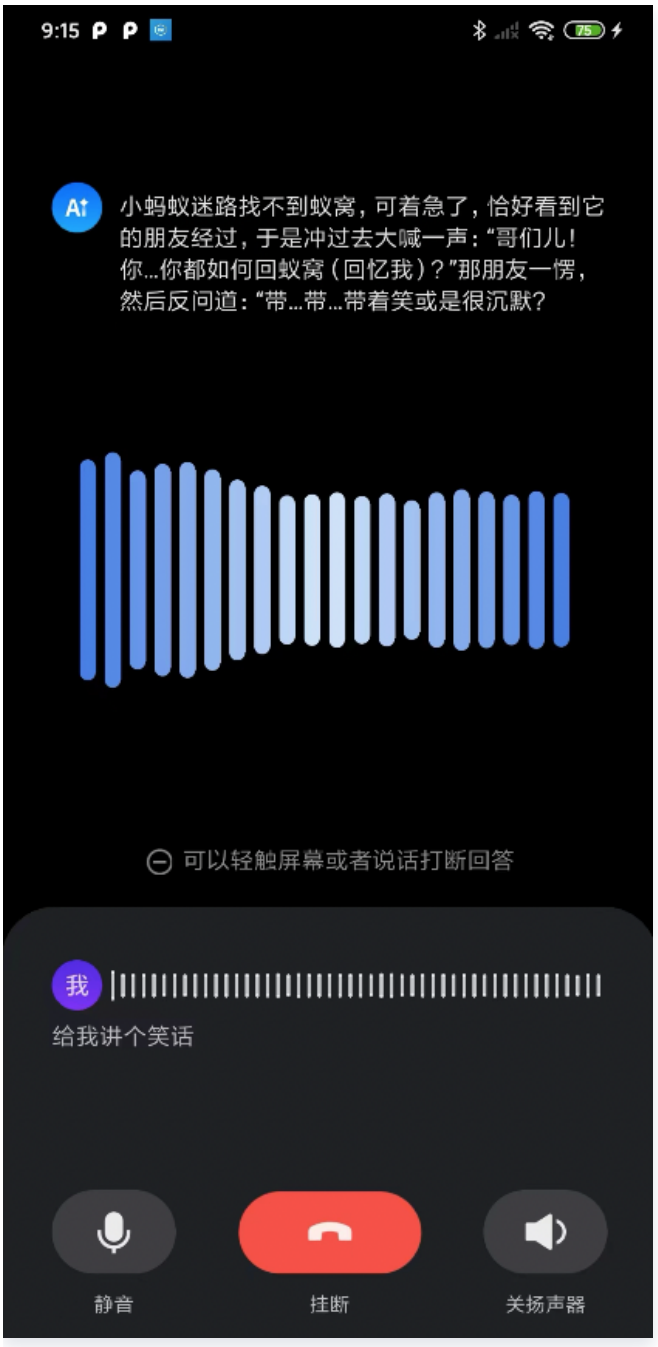
2. 选择可以运行的设备。



3. 使用 `cmd + r` 运行 Demo。

开始您的第一次对话

开始和 AI 说话，比如让 AI 讲个笑话。


9:15
小蚂蚁迷路找不到蚁窝，可着急了，恰好看到它的朋友经过，于是冲过去大喊一声："哥们儿！你...你都如何回蚁窝（回忆我）？"那朋友一愣，然后反问道："带...带...带着笑或是很沉默？
可以轻触屏幕或者说话打断回答
我
给我讲个笑话
静音
挂断
关扬声器

快速集成（无 UI）
快速跑通

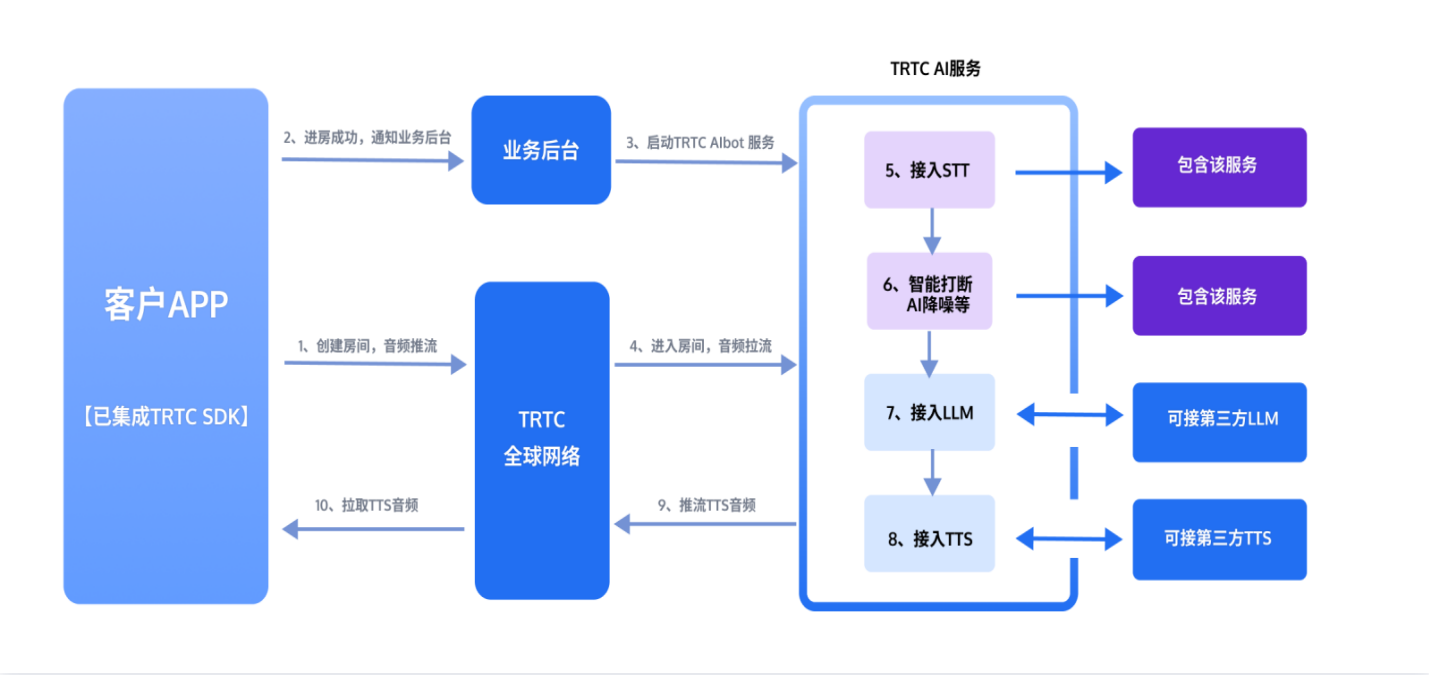
最近更新时间：2025-01-22 17:45:02

本文将介绍如何基于 TRTC SDK 实现 AI 实时对话的解决方案。

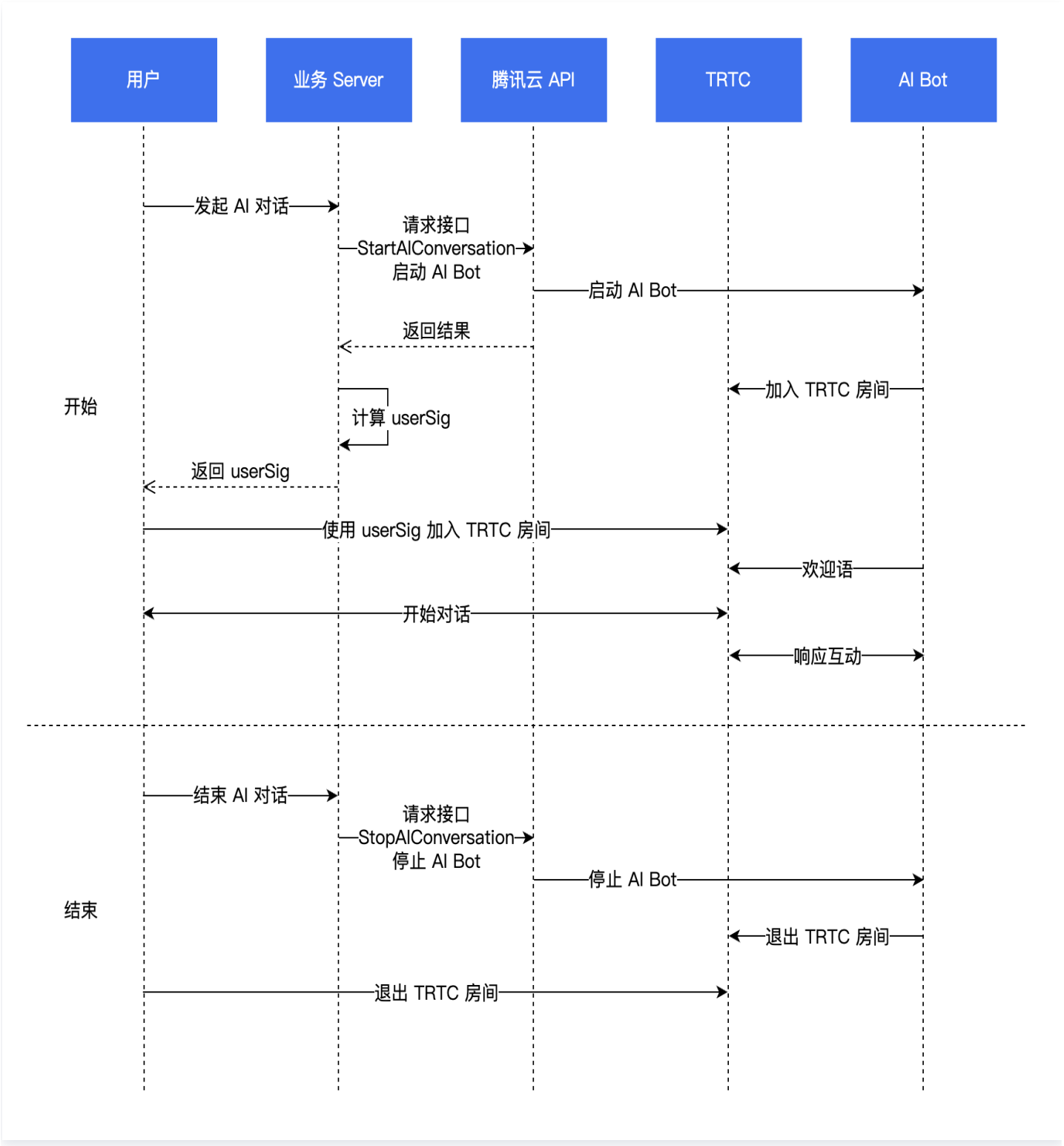
方案介绍

方案基于 TRTC SDK 调用 TRTC 服务，通过调用 AI 实时对话的接口，可以实现极低延迟的 AI 实时对话服务。本方案为您提供了非常灵活的集成方案，您可以根据业务的实际需求接入第三方的 LLM 和 TTS，实现高效的业务实践效果。在整体方案中，我们针对语音实时降噪、AI 智能打断、上下文管理都有较多的技术优化，不断提升用户体验。

方案架构图



业务流程图


用户
业务 Server
腾讯云 API
TRTC
AI Bot
发起 AI 对话
请求接口
StartAIConversation
启动 AI Bot
启动 AI Bot
返回结果
加入 TRTC 房间
开始
计算 userSig
返回 userSig
使用 userSig 加入 TRTC 房间
欢迎语
开始对话
响应互动
结束 AI 对话
请求接口
StopAIConversation
停止 AI Bot
停止 AI Bot
结束
退出 TRTC 房间
退出 TRTC 房间

集成指引

前提条件

> ⚠ **注意：**
> AI实时对话调用会产生使用费用，具体详情请参见 AI 实时对话计费说明 。

1. 创建 TRTC 应用 。
2. 开通 语音转文字 服务。
3. 创建腾讯云 TTS（可选，支持第三方）。
4. 创建 LLM 应用：可以自选适合的大模型厂商进行注册。

一、集成 TRTC SDK

第一步：导入 TRTC SDK 到项目中

- iOS 无 UI 集成指引
- Android 无 UI 集成指引
- Web&H5 无 UI 集成指引
- 小程序 无 UI 集成指引
- Flutter 无 UI 集成指引

第二步：进入 TRTC 房间

- Android&iOS&Windows&Mac 进入TRTC房间
- Web&H5 进入TRTC房间
- 小程序 进入TRTC房间
- Flutter 进入TRTC房间

第三步：发布音频流

Android&iOS&Flutter

您可以调用 startLocalAudio 来开启麦克风采集，该接口需要您通过 quality 参数确定采集模式。虽然这个参数的名字叫做 quality，但并不是说质量越高越好，不同的业务场景有最适合的参数选择（这个参数更准确的含义是 scene）。
AI 对话场景下推荐使用 SPEECH 模式，该模式下的 SDK 音频模块会专注于提炼语音信号，尽最大限度的过滤周围的环境噪音，同时该模式下的音频数据也会获得较好的差质量网络的抵抗能力，因此该模式特别适合于“视频通话”和“在线会议”等侧重于语音沟通的场景。

Android

```
// 开启麦克风采集，并设置当前场景为：语音模式（高噪声抑制能力、强弱网络抗性）
```

```
mCloud.startLocalAudio(TRTCCloudDef.TRTC_AUDIO_QUALITY_SPEECH );
```

iOS&Mac

```
self.trtcCloud = [TRTCCloud sharedInstance];
// 开启麦克风采集，并设置当前场景为：语音模式（高噪声抑制能力、强弱网络抗性）
[self.trtcCloud startLocalAudio:TRTCAudioQualitySpeech];
```

Flutter

```
// 开启麦克风采集，并设置当前场景为：语音模式（高噪声抑制能力、强弱网络抗性）
trtcCloud.startLocalAudio(TRTCAudioQuality.speech);
```

Web&H5

使用 trtc.startLocalAudio() 方法开启麦克风，并发布到房间。

```
await trtc.startLocalAudio();
```

小程序

在进入房间后，调用 getPusherInstance().start() 或者开启自动推流模式即可开始推流。

```
enterRoom(options) {
    this.setData({
        pusher: this.TRTC.enterRoom({
          sdkAppID: 1400xxxxx, // 您的腾讯云账号
          userID: 'trtc-user', //当前进房用户的userID
          userSig: 'xxxxxxx', // 您服务端生成的userSig
          roomID: 1234, // 您进房的房间号，
          enableMic: true, // 进房默认开启音频上行
        }),
    }, () => {
        this.TRTC.getPusherInstance().start() // 开始进行推流
    })
```

```
},
```

说明：
AI 实时对话场景对音频采集端的降噪能力有较高要求，为了获得更好的体验，建议 打开 AI 降噪功能 。此外，我们还有专门针对 AI 实时对话场景训练的降噪模型，欢迎通过商务或 提交工单 联系我们。

二、发起 AI 对话

开始 AI 对话：StartAIConversation

通过业务后台调用 开始 AI 对话任务 接口，拉机器人进房，来发起 AI 实时对话。

注意：
- `RoomId` 需要和客户端进房的 `RoomId` 保持一致，并且房间号的类型（数字房间号、字符串房间号）也必须相同（既机器人和用户需要在同一个房间）。
- `LLMConfig` 和 `TTSConfig` 均为 JSON 字符串，需要正确配置才能成功发起 AI 实时对话。

目前支持的 `LLMConfig` 和 `TTSConfig` 配置说明：
- 大模型配置（LLMConfig）
- 文字转语音配置（TTSConfig）

推荐第一次调用 开始 AI 对话任务 接口前，通过以下页面来验证 `LLMConfig` 和 `TTSConfig` 的参数，具体如下：
- AI 对话参数验证
- AI 对话参数验证（在线地址）

说明：
如果以上步骤都操作正确，现在已经可以和 AI 进行对话啦！

三、接收 AI 对话字幕及 AI 状态

通过 TRTC SDK 接收自定义消息功能，在客户端上监听回调来接收实时字幕与 AI 状态等数据。**cmdID 固定是 1。**

接收实时字幕

消息格式

```
{
  "type": 10000, // 10000表示是下发的实时字幕
```

```
  "sender": "user_a", // 说话人的userid
  "receiver": [], // 接收者userid列表，该消息实际是在房间内广播
  "payload": {
     "text":"", // 语音识别出的文本
     "translation_text":"", // 翻译的文本
     "start_time":"00:00:01", // 这句话的开始时间
     "end_time":"00:00:02", // 这句话的结束时间
     "roundid": "xxxxx", // 唯一标识一轮对话
     "end": true // 如果为true，代表这是一句完整的话
  }
}
```

接收机器人状态

消息格式

```
{
  "type": 10001, // 机器人的状态
  "sender": "user_a", // 发送者userid，这里是机器人的id
  "receiver": [], // 接受者userid列表，该消息实际是在房间内广播
  "payload": {
    "roundid": "xxx", // 唯一标识一轮对话
    "timestamp": 123,
    "state": 1,       //   1 聆听中   2 思考中   3 说话中   4 被打断
  }
}
```

示例代码

Android

```
@Override
public void onRecvCustomCmdMsg(String userId, int cmdID, int seq, byte[]
message) {
    String data = new String(message, StandardCharsets.UTF_8);
    try {
        JSONObject jsonData = new JSONObject(data);
        Log.i(TAG, String.format("receive custom msg from %s cmdId: %d
seq: %d data: %s", userId, cmdID, seq, data));
    } catch (JSONException e) {
        Log.e(TAG, "onRecvCustomCmdMsg err");
```

```
            throw new RuntimeException(e);
        }
    }
```

iOS

```
func onRecvCustomCmdMsgUserId(_ userId: String, cmdID: Int, seq: UInt32,
message: Data) {
    if cmdID == 1 {
        do {
            if let jsonObject = try JSONSerialization.jsonObject(with:
message, options: []) as? [String: Any] {
                print("Dictionary: \(jsonObject)")
                // handleMessage(jsonObject)
            } else {
                print("The data is not a dictionary.")
            }
        } catch {
            print("Error parsing JSON: \(error)")
        }
    }
}
```

Web&H5

```
trtcClient.on(TRTC.EVENT.CUSTOM_MESSAGE, (event) => {
    let data = new TextDecoder().decode(event.data);
    let jsonData = JSON.parse(data);
    console.log(`receive custom msg from ${event.userId} cmdId:
${event.cmdId} seq: ${event.seq} data: ${data}`);

    if (jsonData.type == 10000 && jsonData.payload.end == false) {
        // 字幕中间状态
    } else if (jsonData.type == 10000 && jsonData.payload.end == true) {
       //  一句话说完了
    }
});
```

说明：

> 我们有更多 AI 对话客户端上的回调，具体可参见：AI 对话状态回调 、AI 对话字幕回调 、AI 对话指标回调 、AI 对话错误回调 。

四、发送自定义消息

统一通过端上发送 TRTC 自定义消息，cmdID 固定是2。

参见 发送和接收消息 。

- **可以通过发送自定义的文本，跳过 ASR 过程，直接跟 AI Service 进行文字沟通。**

```
  {
  "type": 20000, // 端上发送自定义文本消息
  "sender": "user_a", // 发送者userid， 服务端会check该userid是否有效
  "receiver": ["user_bot"], // 接受者userid列表，只需要填写机器人userid，服
务端会check该userid是否有效
  "payload": {
    "id": "uuid", // 消息id，可以使用uuid，排查问题使用
    "message": "xxx", // 消息内容
    "timestamp": 123 // 时间戳，排查问题使用
  }
}
```

- **可以通过发送打断信令来进行打断。**

```
{
  "type": 20001, // 端上发送打断信令
  "sender": "user_a", // 发送者userid， 服务端会check该userid是否有效
  "receiver": ["user_bot"], // 接受者userid列表，只需要填写机器人userid，服
务端会check该userid是否有效
  "payload": {
    "id": "uuid", // 消息id，可以使用uuid，排查问题使用
    "timestamp": 123 // 时间戳，排查问题使用
  }
}
```

示例代码

Android

```
public void sendInterruptCode() {
    try {
        int cmdID = 0x2;
```

```
        long time = System.currentTimeMillis();
        String timeStamp = String.valueOf(time/1000);
        JSONObject payLoadContent = new JSONObject();
        payLoadContent.put("timestamp", timeStamp);
        payLoadContent.put("id",
String.valueOf(GenerateTestUserSig.SDKAPPID) + "_" + mRoomId);

        String[] receivers = new String[]{robotUserId};

        JSONObject interruptContent = new JSONObject();
        interruptContent.put("type",
AICustomMsgType.AICustomMsgType_Send_Interrupt_CMD);
        interruptContent.put("sender", mUserId);
        interruptContent.put("receiver", new JSONArray(receivers));
        interruptContent.put("payload", payLoadContent);

        String interruptString = interruptContent.toString();
        byte[] data = interruptString.getBytes("UTF-8");

        Log.i(TAG, "sendInterruptCode :" + interruptString);

        mTRTCCloud.sendCustomCmdMsg(cmdID, data, true, true);
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    } catch (JSONException e) {
        throw new RuntimeException(e);
    }
}
```

iOS

```
@objc func interruptAi() {
    print("interruptAi")
    let cmdId = 0x2
    let timestamp = Int(Date().timeIntervalSince1970 * 1000)
    let payload = [
        "id": userId + "_\(roomId)" + "_\(timestamp)", // 消息id，可以使用
uuid，排查问题使用
        "timestamp": timestamp // 时间戳，排查问题使用
    ] as [String : Any]
```

```
    let dict = [
        "type": 20001,
        "sender": userId,
        "receiver": [botId],
        "payload": payload
    ] as [String : Any]
    do {
        let jsonData = try JSONSerialization.data(withJSONObject: dict,
options: [])
        self.trtcCloud.sendCustomCmdMsg(cmdId, data: jsonData, reliable:
true, ordered: true)
    } catch {
        print("Error serializing dictionary to JSON: \(error)")
    }
}
```

Web&H5

```
const message = {
  "type": 20001,
  "sender": "user_a",
  "receiver": ["user_bot"],
  "payload": {
    "id": "uuid",
    "timestamp": 123
  }
};

trtc.sendCustomMessage({
  cmdId: 2,
  data: new TextEncoder().encode(JSON.stringify(message)).buffer
});
```

> **注意：**
> 目前小程序端暂不支持接收和发送自定义消息，如果您想在小程序端实现接收字幕或发送消息等功能，需要使用 IM 提供的即时通信来实现，可通过商务或 提交工单 联系我们开通 IM 信令通道。

五、停止 AI 对话，退出 TRTC 房间

1. 服务端停止 AI 对话任务。

   通过业务后台调用 停止 AI 对话 接口，拉机器人进房，来发起 AI 实时对话。

2. 客户端退出退出 TRTC 房间，建议参见 退出房间（Android&iOS&Windows&Mac）。

六、其他功能

1、其他服务端接口

- **查询 AI 对话任务状态：** DescribeAIConversation

  可以查询 AI 对话任务状态。有 4 个值：

  1.1 `Idle` 表示任务未开始。

  1.2 `Preparing` 表示任务准备中。

  1.3 `InProgress` 表示任务正在运行。

  1.4 `Stopped` 表示任务已停止，正在清理资源中。
- **更新 AI 对话启动参数：** UpdateAIConversation

  可以在对话过程中，动态更新 TTS 的音色。
- **控制 AI 对话任务：** ControlAIConversation

  当您想让机器人主动播报文本的时候，可以使用该接口。

2、开启服务端回调

可参见 AI 对话服务端回调

> ⚠ **注意：**
> - 回调地址在 TRTC 控制台设置，AI 实时对话回调。
> - 可配合 TRTC 房间与媒体回调 配合使用，丰富功能。

3、其余高级功能介绍

功能	操作指引
智能打断	智能打断逻辑
实现上下文管理	如何实现上下文管理
调用 function call	点击查看示例

大模型配置

最近更新时间：2025-02-18 19:10:02

本文主要介绍如何在 开始AI对话任务 StartAIConversation 接口中配置 `LLMConfig` 参数。

参数说明

名称	描述
LLMType	大模型类型，只要是符合 `openai api` 协议的大模型，都填写 `openai` 。
Model	具体的模型名称，比如 `gpt-4o` 。
APIKey	大模型的 `APIKey` 。
APIUrl	大模型的 `APIUrl` 。
Streaming	是否流式传输。
SystemPrompt	提示词。
Timeout	超时时间，取值范围 [1~50]， 默认为 3 秒（单位：秒）。
History	设置 LLM 的上下文轮次，默认值：0（不提供上下文管理），最大值：50（提供最近 50 轮的上下文管理）。
MetaInfo	自定义参数，会放在请求的 body 中透传给大模型。

LLMConfig 配置示例

openai

```
"LLMConfig": {
      "LLMType": "openai",
      "Model":"gpt-4o",
      "APIKey":"api-key",
      "APIUrl":"https://api.openai.com/v1/chat/completions",
      "Streaming": true,
      "SystemPrompt": "你是一个个人助手",
      "Timeout": 3.0,
      "History": 5,
      "MetaInfo": {}
```

```
}
```

minimax

```
"LLMConfig":{
    "LLMType": "openai",
    "Model": "abab6.5s-chat",
    "Streaming": true,
    "SystemPrompt": "你是一个个人助手",
    "APIKey": "eyJhbGcixxxx",
    "APIUrl": "https://api.minimax.chat/v1/text/chatcompletion_v2",
    "History": 5,
    "MetaInfo": {}
 }
```

混元

```
"LLMConfig":{
      "LLMType": "openai",
      "Model": "hunyuan-standard", // hunyuan-turbo、hunyuan-standard
      "APIKey": "hunyuan-apikey",
      "APIUrl":
"https://hunyuan.cloud.tencent.com/openai/v1/chat/completions",
      "Streaming": true,
      "History": 10,
      "MetaInfo": {}
}
```

说明：
只要是符合 `openai api` 协议的大模型，我们都可以支持， `LLMType` 填写 `openai` ，其他参数根据实际情况填写即可。

大模型的请求

此外我们会在 http header 中增加多个参数来辅助用户支持更复杂的逻辑：

```
X-Task-Id: <task_id_value> // 此任务的 id,
X-Rquest-Id: <request_id> // 此次请求的id， 重试会携带相同的requestId
X-Sdk-App-Id:  SdkAppId
X-User-Id: UserId
```

```
X-Room-Id: RoomId
X-Room-Id-Type: "0" // "0"表示数字房间号 "1"表示字符串房间号
```

文字转语音配置

最近更新时间：2025-01-07 17:56:02

本文主要介绍如何配置 开始AI对话任务 StartAIConversation 接口中的 `TTSConfig` 参数。

支持的 TTSConfig 配置

TTS 参数使用用户自己的第三方账号。

Tencent TTS

```
{
    "TTSType": "tencent", // String TTS类型，目前支持"tencent" 和“minixmax”， 其他的厂商支持中
    "AppId": "您的应用ID", // String 必填
    "SecretId": "您的密钥ID", // String 必填
    "SecretKey":  "您的密钥Key", // String 必填
    "VoiceType": 101001, // Integer  必填，音色 ID，包括标准音色与精品音色，精品音色拟真度更高，价格不同于标准音色，请参见语音合成计费概述。完整的音色 ID 列表请参见语音合成音色列表。
    "Speed": 1.25, // Integer 非必填，语速，范围：[-2，6]，分别对应不同语速：-2：代表0.6倍 -1：代表0.8倍 0：代表1.0倍（默认） 1：代表1.2倍 2：代表1.5倍  6：代表2.5倍  如果需要更细化的语速，可以保留小数点后 2 位，例如0.5/1.25/2.81等。 参数值与实际语速转换，可参考 语速转换
    "Volume": 5, // Integer 非必填，音量大小，范围：[0，10]，分别对应11个等级的音量，默认值为0，代表正常音量。
    "PrimaryLanguage": 1, // Integer 可选 主要语言 1-中文（默认） 2-英文 3-日文
    "FastVoiceType": "xxxx"    //  可选参数， 快速声音复刻的参数
}
```

参见：语音合成 音色列表-文档中心-腾讯云

minimax TTS

```
{
    "TTSType": "minimax",  // String TTS类型,
    "Model": "speech-01-turbo",
    "APIUrl": "https://api.minimax.chat/v1/t2a_v2",
    "APIKey": "eyxxxx",
    "GroupId": "181000000000000",
    "VoiceType":"female-tianmei-jingpin",
```

```
    "Speed": 1.2
}
```

参见：MiniMax-与用户共创智能

限频参见：MiniMax-与用户共创智能 可能会导致回答卡顿

接口名	T2A v2（语音生成）	T2A Pro（语音生成）	T2A（语音生成）	T2A Stream（流式语音生成）	T2A Stream（流式语音生成）
模型	speech-01-turbo、speech-01-240228、speech-01-turbo-240228	speech-01、speech-02	speech-01、speech-02	speech-01	speech-01
客户类型\限制类型	RPM	RPM	RPM	RPM	CONN（最大并行运行任务数）
免费用户	3	3	3	3	1
充值用户	20	20	20	20	3

Azure TTS

```
{
    "TTSType": "azure", // 必填：String TTS类型
    "SubscriptionKey": "xxxxxxxx", // 必填：String 订阅的Key
    "Region": "chinanorth3",  // 必填：String 订阅的地区
    "VoiceName": "zh-CN-XiaoxiaoNeural", // 必填：String 音色名必填
    "Language": "zh-CN", // 必填：String 合成的语言
    "Rate": 1 // 选填：float 语速  0.5~2 默认为 1
}
```

参见：使用 SSML 来自定义语音和声音

Cartesia TTS

```
{
    "TTSType": "cartesia",  // 必填:String TTS类型,
```

```
    "Model": "sonic-multilingual", //必填 模型
    "APIKey": "eyxxxx",//必填：获取的api密钥
    "VoiceId": "eda5bbff-1ff1-4886-8ef1-4e69a77640a0"  //必填 声音id
https://play.cartesia.ai/
}
```

参见：Cartesia TTS

ElevenLabs TTS

```
{
    "TTSType": "elevenlabs",  // 必填:String TTS类型,
    "Model": "eleven_turbo_v2_5",//必填:模型类型
    "APIKey": "eyxxxx",
    "VoiceId": "eda5bbff-1ff1-4886-8ef1-4e69a77640a0"  //声音类型
https://elevenlabs.io/docs/api-reference/get-voices
}
```

参见：ElevenLabs TTS

自定义 TTS

```
{
  "TTSType": "custom", // String 必填
  "APIKey": "ApiKey", // String 必填 用来鉴权
  "APIUrl": "http://0.0.0.0:8080/stream-audio" // String，必填，TTS API
URL
  "AudioFormat": "wav", // String, 非必填，期望输出的音频格式，如mp3，
ogg_opus，pcm，wav，默认为 wav，目前只支持pcm和wav，
  "SampleRate": 16000,  // Integer，非必填，音频采样率，默认为16000(16k)，推荐
值为16000
  "AudioChannel": 1,    // Integer，非必填，音频通道数，取值：1 或 2  默认为1
}
```

具体协议规范参见 自定义 TTS 协议 。

智能打断逻辑

最近更新时间：2024-11-08 18:05:32

智能打断概述

智能打断是实时语音对话中至关重要的功能，直接影响用户体验。TRTC AI 对话凭借其丰富的打断方案，为客户提供了灵活而强大的解决方案，满足各种场景需求。我们的智能打断功能主要分为自动打断和手动打断两种模式，能够有效提升对话的流畅性和自然度。

自动打断

自动打断是指系统在检测到特定条件时，根据用户的音频输入自动触发，支持两种主要规则：基于音频时长的打断和基于语义的打断。

1. **基于音频时长的打断**
   - 用户可动态配置打断时长，默认为500ms。
   - 可调范围：300ms（更灵敏）到5000ms（避免误打断）。
   - 建议：
     - 对于客服对话等高交互场景，设置较短时长（如300ms 或者 500ms）以提高响应速度。
     - 对于演讲或授课场景，设置较长时长（如1000ms）以避免误打断。
2. **基于语义的打断**

   基于语音活动检测（VAD）技术，当服务端检测到用户输入了一句语义完整的话时，会自动触发打断。
3. **自动打断与手动打断的配合**

   在自动打断模式下，仍可使用端上 SDK 发送自定义打断信令，实现特定的手动打断控制，提供更精细的交互体验。

手动打断

手动打断手动打断是指由用户主动发起，通过端上 SDK 发送自定义消息实现，为用户提供更直接的控制方式。通常依赖于用户界面上的按钮或快捷键。

1. **打断信令说明**

   手动打断使用如下 JSON 格式的自定义消息： 统一通过端上发送 TRTC 自定义消息，**cmdID 固定是2**。

```
{
  "type": 20001, // 固定type
  "sender": "user_a", // 发送者userid,服务端会验证有效性
  "receiver": ["user_bot"], // 接收者userid列表,填写机器人userid
  "payload": {
    "id": "uuid", // 消息id,用于问题排查
    "timestamp": 123 // 时间戳,用于问题排查
```

```
    }
}
```

2. **使用方法**

   通过 SDK 的 `sendCustomCmdMessage` 方法发送上述自定义消息即可实现手动打断。

应用场景示例

1. 客服对话：使用较短的自动打断时长，配合手动打断，实现快速响应和精准控制。
2. 在线教育：采用较长的自动打断时长，避免误打断学生发言，教师可使用手动打断进行必要干预。
3. 多人会议：结合自动打断和手动打断，灵活管理发言顺序和时长。

如何实现上下文管理

最近更新时间：2024-11-08 18:05:32

大模型的上下文管理非常重要，它可以让大模型根据聊天历史进行更精准的内容回答。TRTC AI 提供了基本的上下文管理能力，同时也支持开发者实现自己的丰富上下文管理方案。

基本的上下文管理

TRTC AI 提供了基本的上下文管理功能。在 LLMConfig 参数中，我们引入了 History 参数来控制上下文管理：
History：

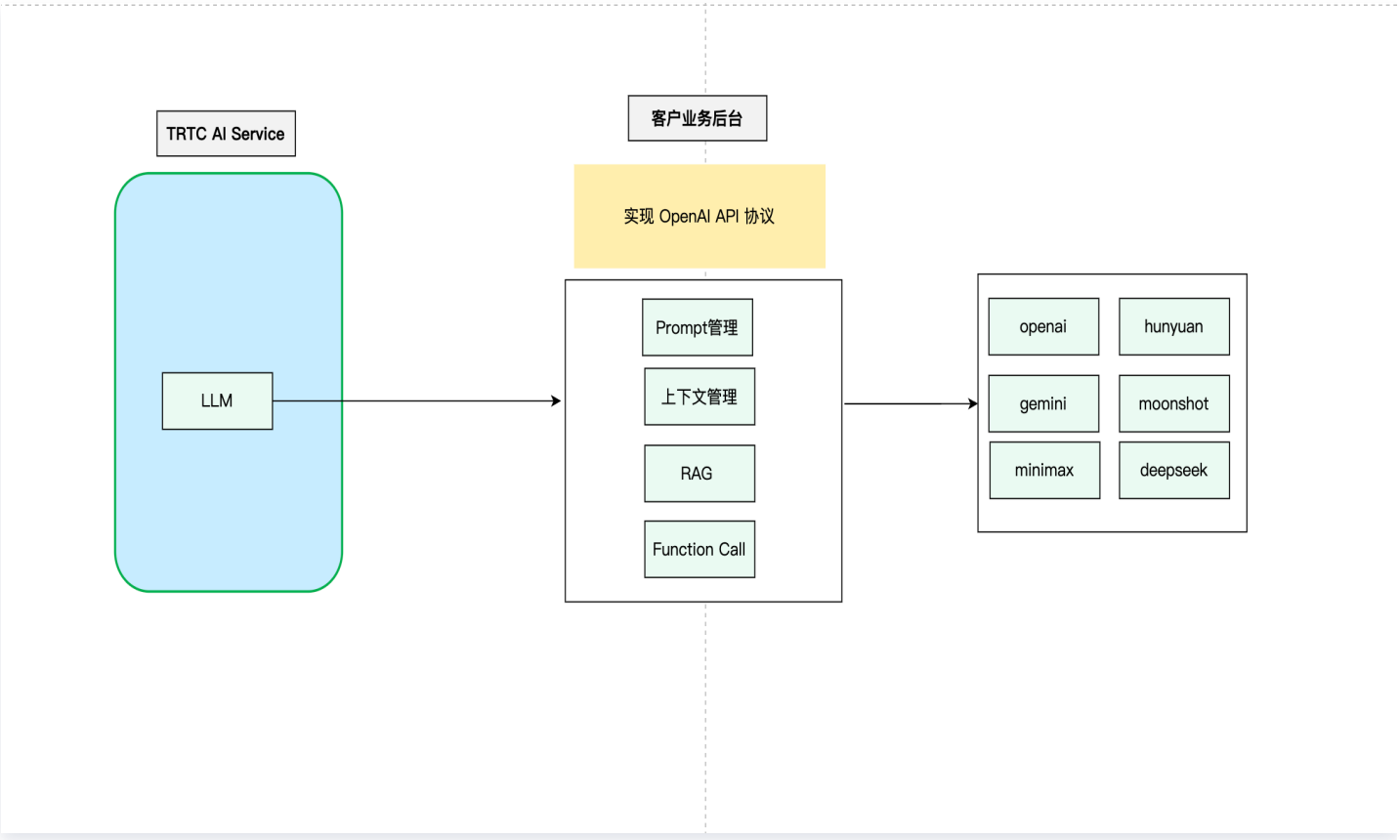
- 设置 LLM 的上下文轮次默认值0（不提供上下文管理）。
- 最大值：50（提供最近50轮的上下文管理）。

相关配置示例如下：

```
"LLMConfig": {
          "LLMType": "openai",
          "Model":"gpt-4o",
          "APIKey":"api-key",
          "APIUrl":"https://api.openai.com/chat/completions",
          "Streaming": true,
          "SystemPrompt": "你是一个个人助手",
          "Timeout": 3.0,
          "History": 5     //  最大支持 50 轮对话，默认为 0
  }
```

自定义上下文管理

TRTC AI 对话服务支持标准的 OpenAI 规范，这使得开发者能够在自己的业务中实现定制化的上下文管理。实现流程如下：



这个流程图展示了自定义上下文管理的基本步骤。开发者可以根据自己的具体需求对这个流程进行调整和优化。

实现示例

开发者可以在自己的业务后台实现与 OpenAI API 兼容的大模型接口，并将封装了上下文逻辑的大模型请求发送给第三方大模型。以下是一个简化的示例代码:

```
import time
from fastapi import FastAPI, HTTPException
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel
from typing import List, Optional
from langchain_core.messages import HumanMessage, SystemMessage
from langchain_openai import ChatOpenAI


app = FastAPI(debug=True)

# 添加 CORS 中间件
app.add_middleware(
    CORSMiddleware,
```

```
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)


class Message(BaseModel):
    role: str
    content: str


class ChatRequest(BaseModel):
    model: str
    messages: List[Message]
    temperature: Optional[float] = 0.7


class ChatResponse(BaseModel):
    id: str
    object: str
    created: int
    model: str
    choices: List[dict]
    usage: dict


@app.post("/v1/chat/completions")
async def chat_completions(request: ChatRequest):
    try:
        # 将请求消息转换为 LangChain 消息格式
        langchain_messages = []
        for msg in request.messages:
            if msg.role == "system":

langchain_messages.append(SystemMessage(content=msg.content))
            elif msg.role == "user":

langchain_messages.append(HumanMessage(content=msg.content))


        # add more historys
```

```
        # 使用 LangChain 的 ChatOpenAI 模型
        chat = ChatOpenAI(temperature=request.temperature,
                          model_name=request.model)
        response = chat(langchain_messages)
        print(response)

        # 构造符合 OpenAI API 格式的响应
        return ChatResponse(
            id="chatcmpl-" + "".join([str(ord(c))
                                      for c in response.content[:8]]),
            object="chat.completion",
            created=int(time.time()),
            model=request.model,
            choices=[{
                "index": 0,
                "message": {
                    "role": "assistant",
                    "content": response.content
                },
                "finish_reason": "stop"
            }],
            usage={
                "prompt_tokens": -1,  # LangChain 不提供这些信息，所以我们使用
占位值
                "completion_tokens": -1,
                "total_tokens": -1
            }
        )
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

详情请点击 查看示例 。

AI 对话 SDK 端回调
AI 对话状态回调

最近更新时间：2025-01-08 10:03:32

TRTC AI 对话提供了丰富的状态回调功能。这些状态回调通过 TRTC 的自定义消息发送，使得在客户端上可以方便地实现丰富的状态切换，例如"聆听中"、"思考中"、“说话中”等状态。
通过 TRTC SDK 接收自定义消息功能，在客户端上监听回调来接收实AI 状态的数据。**cmdID 固定是1。**

功能特点

1. 实时性：状态变化可以实时传递给所有参与者。
2. 灵活性：使用自定义消息格式，便于集成和扩展。
3. 多状态支持：包括聆听、思考、说话和被打断等多种状态。

消息格式

状态回调消息使用 JSON 格式，具体字段如下：

字段	类型	描述
type	Number	消息类型，10001 表示机器人状态
sender	String	发送者的 userid，这里是机器人的 id
receiver	Array	接收者 userid 列表，该消息实际在房间内广播
payload	Object	消息负载，包含状态详细信息

payload 对象包含以下字段：

字段	类型	描述
roundid	String	唯一标识一轮对话的 ID
timestamp	Number	时间戳，表示状态变化的具体时间
state	Number	机器人当前状态代码

状态代码说明：

状态代码	描述
1	聆听中

2	思考中
3	说话中
4	被打断

示例消息

```
{
  "type": 10001,
  "sender": "ai_assistant_001",
  "receiver": [],
  "payload": {
    "roundid": "conversation_789012",
    "timestamp": 1629384755,
    "state": 2
  }
}
```

应用场景

1. UI 反馈：根据不同状态显示相应的界面元素，如“正在聆听”的动画。
2. 交互控制：在 AI 思考或说话时，可以禁用某些用户输入以避免中断。
3. 调试与分析：利用状态信息进行会话分析，优化用户体验。

AI 对话字幕回调

最近更新时间：2025-01-08 10:03:32

TRTC AI 对话提供了显示实时字幕的能力。实时字幕通过 TRTC 的自定义消息发送，可以实现与音频对话毫秒级别的同步。
通过 TRTC SDK 接收自定义消息功能，在客户端上监听回调来接收实AI 字幕的数据。**cmdID 固定是1。**

功能特点

1. 实时性：字幕与音频对话同步，延迟在毫秒级别。
2. 灵活性：使用自定义消息格式，便于集成和扩展。

消息格式

实时字幕消息使用 JSON 格式，具体字段如下：

字段	类型	描述
type	Number	消息类型，10000表示实时字幕
sender	String	说话人的 userid
receiver	Array	接收者 userid 列表，该消息实际在房间内广播
payload	Object	消息负载，包含字幕详细信息

payload 对象包含以下字段：

字段	类型	描述
text	String	语音识别出的原文本
start_time	String	这句话的开始时间，格式为 "HH:MM:SS"
end_time	String	这句话的结束时间，格式为 "HH:MM:SS"
roundid	String	唯一标识一轮对话的 ID
end	Boolean	如果为 true，代表这是一句完整的话

示例消息

```
{
  "type": 10000,
```

```
  "sender": "user_a",
  "receiver": [],
  "payload": {
    "text": "你好，很高兴认识你。",
    "start_time": "00:00:01",
    "end_time": "00:00:03",
    "roundid": "conversation_123456",
    "end": true
  }
}
```

实现注意事项

1. 消息处理：接收方需要正确解析 JSON 消息，并根据 type 字段识别实时字幕消息。
2. 时间同步：使用 start_time 和 end_time 确保字幕与音频正确对齐。
3. 对话分段：使用 end 字段判断一句话是否结束，可用于界面更新或存储完整对话。

Web SDK 自定义消息解析如下

```
    trtcClient.on(TRTC.EVENT.CUSTOM_MESSAGE, (event) => {
        let data = new TextDecoder().decode(event.data);
        let jsonData = JSON.parse(data);
        console.log(`receive custom msg from ${event.userId} cmdId:
${event.cmdId} seq: ${event.seq} data: ${data}`);

        if (jsonData.type == 10000 && jsonData.payload.end == false) {
            // 字幕中间状态
        } else if (jsonData.type == 10000 && jsonData.payload.end ==
true) {
            //  一句话说完了
        }
    });
```

AI 对话指标回调

最近更新时间：2025-01-08 10:03:32

TRTC AI 对话提供了丰富的指标回调功能。这些状态回调通过 TRTC 的自定义消息发送，使得在客户端上可以方便监控对话运行情况，如 LLM、TTS 调用耗时以及性能数据通过 TRTC 自定义消息实时推送到终端，监控 AI 对话的耗时与质量等。
通过 TRTC SDK 接收自定义消息功能，在客户端上监听回调来接收实AI 对话指标的数据。**cmdID 固定是1。**

字段	类型	描述
type	Number	消息类型，10020 表示AI服务的调用回调
sender	String	发送者的 userid，这里是机器人的 id
payload	Object	消息负载，包含指标详细信息

payload 对象包含以下字段：

字段	类型	描述
metric	String	调用指标名称如下： • asr_latency • llm_network_latency • llm_first_token • tts_network_latency • tts_first_frame_latency • tts_discontinuity • interruption
value	Number	调用指标
tag	Object	指标关联的 tag

tag 对象包含以下字段：

字段	类型	描述
roundid	String	对话轮次 ID

指标名称说明：

状态代码	描述

asr_latency	asr 延迟。注意：指标包含启动 AI 对话时 VadSilenceTime 所设置的时间
llm_network_latency	llm 请求的网络耗时
llm_first_token	llm 首 token 耗时，指标包含网络耗时
tts_network_latency	tts 请求的网络耗时
tts_first_frame_latency	tts 首帧耗时，指标包含网络耗时
tts_discontinuity	tts 未连续的次数，代表 tts 流式请求播放完成之后，下一个请求还没有返回结果，通常是 tts 延迟比较高导致
interruption	表示此轮对话被打断

AI 对话错误回调

最近更新时间：2025-01-08 10:03:32

TRTC AI 对话提供了错误回调功能。这些状态回调通过 TRTC 的自定义消息发送，使得在客户端上可以方便监控对话运行情况，如 LLM、TTS 调用失败时会将数据通过 TRTC 自定义消息实时推送到终端，监控 AI 对话的质量与成功率等。
通过 TRTC SDK 接收自定义消息功能，在客户端上监听回调来接收实AI 对话错误回调的数据。**cmdID 固定是 1。**

字段	类型	描述
type	Number	消息类型，10030 表示 AI 服务的错误回调
sender	String	发送者的 userid，这里是机器人的 id
payload	Object	消息负载，包含详细信息

payload 对象包含以下字段：

字段	类型	描述
metric	String	调用指标名称如下 • asr_error • llm_error • tts_error
tag	Object	指标关联的 tag

tag 对象包含以下字段：

字段	类型	描述
roundid	String	对话轮次 ID
code	Number	调用错误码
message	String	错误信息的详细描述

指标名称说明：

状态代码	描述
asr_error	asr 识别出现错误

llm_error	llm 请求出现错误
tts_error	tts 请求出现错误

错误码列表

服务类别	错误码	错误描述
ASR	30100	请求超时
	30102	内部错误
LLM	30200	请求 LLM 超时
	30201	LLM 请求被频率限制
	30202	LLM 服务返回失败
TTS	30300	请求 TTS 服务超时
	30301	TTS 请求被频率限制
	30302	TTS 服务返回失败

说明：
详细的错误码，例如第三方 LLM、TTS 的错误码会包装在 `tag.message` 中，便于快速定位问题以及提示用户出错。

AI 对话服务端回调

最近更新时间：2025-02-25 18:15:22

本文用于介绍 AI 服务（ AI 实时对话 ）相关的云 API 接口产生的事件，以 HTTP/HTTPS 请求的形式通知到您的服务器。您可以向腾讯云提供相关的配置信息来开通该服务。您也可以结合TRTC的 房间与媒体回调 使用，实现更多自定义逻辑。

配置信息

实时音视频 TRTC 控制台支持自助配置回调信息，配置完成后即可接收事件回调通知。详细操作指引请参见 回调配置 。

> **注意：**
> 您需要提前准备以下信息：
> - **必要项**：接收回调通知的 HTTP/HTTPS 服务器地址。
> - **可选项**：计算签名的 密钥 key ，由您自定义一个最大32个字符的 key，以大小写字母及数字组成。

超时重试

事件回调服务器在发送消息通知后，5秒内没有收到您的服务器的响应，即认为通知失败。首次通知失败后会立即重试，后续失败会以10秒的间隔继续重试，直到消息存续时间超过1分钟，不再重试。

事件回调消息格式

事件回调消息以 HTTP/HTTPS POST 请求发送给您的服务器，其中：

- **字符编码格式**：UTF-8。
- **请求**：body 格式为 JSON。
- **应答**：HTTP STATUS CODE = 200，服务端忽略应答包具体内容，为了协议友好，建议客户应答内容携带 JSON： {"code":0}。
- **包体示例**：下述为“启动AI对话任务成功”事件的包体示例。

```
{
    "EventGroupId":  9,
    "EventType": 901,
    "CallbackTs": 1687770730166,
    "EventInfo": {
        "EventMsTs": 1622186275757,
        "TaskId": "hKPD2Q7kBVzu-
6ezFiqmcEBJQCykqbZrS9OOTE46uYlb4NvQDIaEXlpOlLXFtGBiado5oP0zfLDZs",
        "RoomId": "1234",
```

```
        "RoomIdType": 0,
        "Payload": {
            "Status": 0
        }
    }
}
```

参数说明

回调消息参数

- 事件回调消息的 header 中包含以下字段：

字段名	值
Content-Type	application/json
Sign	签名值
SdkAppId	sdk application id

- 事件回调消息的 body 中包含以下字段：

字段名	类型	含义
EventGroupId	Number	事件组 ID，混流转推事件固定为4
EventType	Number	回调通知的事件类型
CallbackMsTs	Number	事件回调服务器向您的服务器发出回调请求的 Unix 时间戳，单位为毫秒
EventInfo	JSON Object	事件信息

事件组 ID

字段名	值	含义
EVENT_GROUP_AI_SERVICE	9	AI 服务事件组

事件类型

字段名	值	含义

EVENT_TYPE_AI_SERVICE_START	901	AI 任务开始状态回调
EVENT_TYPE_AI_SERVICE_STOP	902	AI 任务结束状态回调
EVENT_TYPE_AI_SERVICE_MSG	903	回调ASR识别的完整一句话或 LLM 返回的完整内容
EVENT_TYPE_AI_START_OF_SPEECH	904	ASR 识别一句话的开始回调
EVENT_TYPE_AI_SPEAKING_FINISHED	905	在一轮对话中，AI 说完话的回调
EVENT_TYPE_AI_METRIC_MESSAGE	906	AI 服务中调用 LLM/TTS 的指标回调
EVENT_TYPE_AI_ERROR_METRIC_CALLBACK	908	AI 服务的调用指标错误回调
EVENT_TYPE_AI_SESSION_STATUS_CALLBACK	909	AI 会话准备就绪回调，表示音视频通道已建立，可以进行对话。

事件类型为（901）时事件信息的定义：

字段名	类型	含义
EventMsTs	String	事件发生的 Unix 时间戳，单位为毫秒
TaskId	String	AI 任务 ID
RoomId	String	TRTC 的房间 ID
RoomIdType	Integer	• 0：表示数字房间号 • 1：表示字符串房间号
Payload.Status	Number	• 0：启动 AI 任务成功 • 1：启动 AI 任务失败

```
{
    "EventGroupId": 9,
    "EventType": 901,
    "CallbackTs": 1687770730166,
    "EventInfo": {
```

```
        "EventMsTs": 1622186275757,
        "TaskId": "xx",
        "RoomId": "1234",
        "RoomIdType": 0,
        "Payload": {
            "Status": 0
        }
    }
}
```

事件类型为（902）时事件信息的定义：

字段名	类型	含义
EventMsTs	String	事件发生的 Unix 时间戳，单位为毫秒
TaskId	String	AI 任务 ID
RoomId	String	TRTC 的房间 ID
RoomIdType	Integer	• 0：表示数字房间号 • 1：表示字符串房间号
Payload.LeaveCode	Integer	• 0：正常调用停止接口后任务退出 • 1：业务自己踢掉转录机器人后任务退出 • 2：业务自己解散房间后任务退出 • 3：TRTC 服务端踢掉机器人 • 4：TRTC 服务端解散房间 • 98：内部异常错误，建议业务进行重试 • 99：代表房间内除了转录机器人没有其他用户流，超过指定时间退出

```
{
    "EventGroupId": 9,
    "EventType": 902,
    "CallbackTs": 1687770730166,
    "EventInfo": {
        "EventMsTs": 1622186275757,
        "TaskId": "xx",
        "RoomId": "1234",
        "RoomIdType": 0,
        "Payload": {
```

```
            "LeaveCode": 0
        }
    }
}
```

事件类型为（903）时事件信息的定义：

字段名	类型	含义
EventMsTs	String	事件发生的 Unix 时间戳，单位为毫秒
TaskId	String	AI 任务 ID
RoomId	String	TRTC 的房间 ID
RoomIdType	Integer	• 0：表示数字房间号 • 1：表示字符串房间号

Payload	JSON Object	为 JSON 对象，与客户端自定义消息回调格式一致
		{
		"UserId":"",
		"Text":"",
		"StartTimeMs":1234,
		"EndTimeMs":1269,
		"RoundId":"xxxxxx" // 一轮对话的唯一id
		}

```
{
    "EventGroupId":  9,
    "EventType":     903,
    "CallbackTs":    1687770730166,
    "EventInfo": {
        "EventMsTs": 1622186275757,
       "TaskId": "xx",
        "RoomId": "1234",
        "RoomIdType": 0,
        "Payload": {
            "UserId":"",
            "Text":"",
            "StartTimeMs":1234,
            "EndTimeMs":1269,
            "RoundId":"xxxxxx"
        }
```

```
    }
}
```

事件类型为（904）时事件信息的定义：

字段名	类型	含义
EventMsTs	String	事件发生的 Unix 时间戳，单位为毫秒
TaskId	String	AI 任务 ID
RoomId	String	TRTC 的房间 ID
RoomIdType	Integer	• 0：表示数字房间号 • 1：表示字符串房间号

Payload	JSON Object	为 JSON 对象
		{
		"UserId": "xxx",
		"RoundId": "xxxxx" // 一轮对话的唯一id
		}

```
{
    "EventGroupId":  9,
    "EventType":     904,
    "CallbackTs":    1687770730166,
    "EventInfo": {
        "EventMsTs": 1622186275757,
        "TaskId": "xx",
        "RoomId": "1234",
        "RoomIdType": 0,
        "Payload": {
           "UserId": "xxx",
           "RoundId": "xxxxx"
        }
    }
}
```

事件类型为（905）时事件信息的定义：

字段名	类型	含义
EventMsTs	String	事件发生的 Unix 时间戳，单位为毫秒

TaskId	String	AI 任务 ID
RoomId	String	TRTC 的房间 ID
RoomIdType	Integer	• 0：表示数字房间号 • 1：表示字符串房间号

Payload	JSON Object	为 JSON 对象
		{
		"UserId": "UserId"，// AI机器人的UserId
		"RoundId": "RoundId"，// 当前对话的RoundId
		"Text": "Text" // 在本轮对话中，AI机器人已经说的文本
		}

```
{
    "EventGroupId":  9,
    "EventType":     905,
    "CallbackTs":    1687770730166,
    "EventInfo": {
        "EventMsTs": 1622186275757,
        "TaskId": "xx",
        "RoomId": "1234",
        "RoomIdType": 0,
        "Payload": {
            "UserId": "UserId",
            "RoundId": "RoundId",
            "Text": "Text"
        }
    }
}
```

事件类型为（906）时事件信息的定义：

字段名	类型	含义
EventMsTs	String	事件发生的 Unix 时间戳，单位为毫秒
TaskId	String	AI 任务 ID
RoomId	String	TRTC 的房间 ID
RoomIdType	Integer	• 0：表示数字房间号 • 1：表示字符串房间号

Payload	JSON Object	为 JSON 对象
		{
		"Metric": "llm_network_latency", // 指标名
		"Value": 218, // 指标
		"Tag": {
		"RoundId": "070c4908-105", // 对话轮次id
		}
		}
		
		调用指标名称如下：
		• asr_latency
		• llm_network_latency
		• llm_first_token
		• tts_network_latency
		• tts_first_frame_latency
		• tts_discontinuity
		• interruption

```
{
    "EventGroupId":  9,
    "EventType":     906,
    "CallbackTs":    1687770730166,
    "EventInfo": {
        "EventMsTs": 1622186275757,
        "TaskId": "xx",
        "RoomId": "1234",
        "RoomIdType": 0,
        "Payload": {
            "Metric": "llm_first_token",
            "Value": 218,
            "Tag": {
                "RoundId": "070c4908-1057-4ced-a949-356bf11848bc"
            }
        }
    }
}
```

事件类型为（908）时事件信息的定义：

字段名	类型	含义

EventMsTs	String	事件发生的 Unix 时间戳，单位为毫秒
TaskId	String	AI 任务 ID
RoomId	String	TRTC 的房间 ID
RoomIdType	Integer	• 0：表示数字房间号 • 1：表示字符串房间号

Payload	JSON Object	为 JSON 对象
		{
		"Metric": "llm_error", // 指标名
		"Tag": {
		"RoundId": "070c4908-1057-4ced-a949-356bf11848bc",
		"Code": 0, // 服务错误码
		"Message": "" // 错误信息的详细描述
		}
		}
		
		错误指标名称下：
		• asr_error
		• llm_error
		• tts_error

```
{
    "EventGroupId":  9,
    "EventType":     908,
    "CallbackTs":    1687770730166,
    "EventInfo": {
        "EventMsTs": 1622186275757,
        "TaskId": "xx",
        "RoomId": "1234",
        "RoomIdType": 0,
        "Payload": {
            "Metric": "llm_error",
            "Tag":  {
                "RoundId": "070c4908-1057-4ced-a949-356bf11848bc",
                "Code": 0,
                "Message": ""
            }
        }
    }
```

```
}
```

事件类型为（909）时事件信息的定义：

字段名	类型	含义
EventMsTs	String	事件发生的 Unix 时间戳，单位为毫秒
TaskId	String	AI 任务 ID
RoomId	String	TRTC 的房间 ID
RoomIdType	Integer	• 0：表示数字房间号 • 1：表示字符串房间号

Payload	JSON Object	为 JSON 对象
		{
		"Status": "session_ready" // 表示音视频通道已建立，可以进行对话。
		}

```
{
    "EventGroupId":  9,
    "EventType":     909,
    "CallbackTs":    1687770730166,
    "EventInfo": {
        "EventMsTs": 1622186275757,
        "TaskId": "xx",
        "RoomId": "1234",
        "RoomIdType": 0,
        "Payload": {
            "Status": "session_ready"
        }
    }
}
```

计算签名

签名由 HMAC SHA256 加密算法计算得出，您的事件回调接收服务器收到回调消息后，通过同样的方式计算出签名，相同则说明是腾讯云的实时音视频的事件回调，没有被伪造。签名的计算如下所示：

```
//签名 Sign 计算公式中 key 为计算签名 Sign 用的加密密钥。
```

```
Sign = base64(hmacsha256(key, body))
```

注意：

body 为您收到回调请求的原始包体，不要做任何转化，示例如下：

```
body="{\n\t\"Ebody="
{\"EventGroupId\":7,\"EventType\":701,\"CallbackMsTs\":17019379000
12,\"EventInfo\":
{\"EventMsTs\":1701937900012,\"TaskId\":\"WMdqEeEgj2ksqnyUsuXC+qLk
VypGmwjrgh1JC6ZefVP+rvsidDnZsAw8uWgX0XRGvdSVfAMunise2kcZaefdgHvx3-
M2v6fmTjRNgg..\",\"Status\":0}}"ventGroupId\":\t1,\n\t\"EventType\
":\t103,\n\t\"CallbackTs\":\t1615554923704,\n\t\"EventInfo\":\t{\n
\t\t\"RoomId\":\t12345,\n\t\t\"EventTs\":\t1608441737,\n\t\t\"User
Id\":\t\"test\",\n\t\t\"UniqueId\":\t1615554922656,\n\t\t\"Role\":
\t20,\n\t\t\"Reason\":\t1\n\t}\n}"
```

签名校验示例

Java

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
//# 功能：第三方回调sign校验
//# 参数：
//#    key：控制台配置的密钥key
//#    body：腾讯云回调返回的body体
//#    sign：腾讯云回调返回的签名值sign
//# 返回值：
//#    Status：OK 表示校验通过，FAIL 表示校验失败，具体原因参考Info
//#    Info：成功/失败信息

public class checkSign {
    public static String getResultSign(String key, String body) throws
Exception {
        Mac hmacSha256 = Mac.getInstance("HmacSHA256");
        SecretKeySpec secret_key = new SecretKeySpec(key.getBytes(),
"HmacSHA256");
        hmacSha256.init(secret_key);
```

```
        return
Base64.getEncoder().encodeToString(hmacSha256.doFinal(body.getBytes()));
    }
    public static void main(String[] args) throws Exception {
        String key = "123654";
        String body = "{\n" + "\t\"EventGroupId\":\t2,\n" +
"\t\"EventType\":\t204,\n" + "\t\"CallbackTs\":\t1664209748188,\n" +
"\t\"EventInfo\":\t{\n" + "\t\t\"RoomId\":\t8489,\n" +
"\t\t\"EventTs\":\t1664209748,\n" +
"\t\t\"EventMsTs\":\t1664209748180,\n" +
"\t\t\"UserId\":\t\"user_85034614\",\n" + "\t\t\"Reason\":\t0\n" +
"\t}\n" + "}";
        String Sign = "kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA=";
        String resultSign = getResultSign(key, body);

        if (resultSign.equals(Sign)) {
            System.out.println("{'Status': 'OK', 'Info': '校验通过'}");
        } else {
            System.out.println("{'Status': 'FAIL', 'Info': '校验失败'}");
        }
    }
}
```

Python

```
# -*- coding: utf8 -*-
import hmac
import base64
from hashlib import sha256

# 功能：第三方回调sign校验
# 参数：
#   key：控制台配置的密钥key
#   body：腾讯云回调返回的body体
#   sign：腾讯云回调返回的签名值sign
# 返回值：
#   Status：OK 表示校验通过，FAIL 表示校验失败，具体原因参考Info
#   Info：成功/失败信息

def checkSign(key, body, sign):
    temp_dict = {}
```

```
    computSign = base64.b64encode(hmac.new(key.encode('utf-8'),
body.encode('utf-8'), digestmod=sha256).digest()).decode('utf-8')
    print(computSign)
    if computSign == sign:
        temp_dict['Status'] = 'OK'
        temp_dict['Info'] = '校验通过'
        return temp_dict
    else:
        temp_dict['Status'] = 'FAIL'
        temp_dict['Info'] = '校验失败'
        return temp_dict

if __name__ == '__main__':
    key = '123654'
    body = "{\n" + "\t\"EventGroupId\":\t2,\n" +
"\t\"EventType\":\t204,\n" + "\t\"CallbackTs\":\t1664209748188,\n" +
"\t\"EventInfo\":\t{\n" + "\t\t\"RoomId\":\t8489,\n" +
"\t\t\"EventTs\":\t1664209748,\n" +
"\t\t\"EventMsTs\":\t1664209748180,\n" +
"\t\t\"UserId\":\t\"user_85034614\",\n" + "\t\t\"Reason\":\t0\n" +
"\t}\n" + "}"
    sign = 'kkoFeO3Oh2ZHnjtg8tEAQhtXK16/KI05W3BQff8IvGA='
    result = checkSign(key, body, sign)
    print(result)
```

PHP

```
<?php

class TlsEventSig {

    private $key = false;
    private $body = false;

    public function __construct( $key, $body ) {
        $this->key = $key;
        $this->body = $body;
    }

    private function __hmacsha256() {
        $hash = hash_hmac( 'sha256', $this->body, $this->key, true );
        return base64_encode( $hash);
```

```
    }

    public function genEventSig() {
        return $this->__hmacsha256();
    }
}

$key="789";
$data="
{\n\t\"EventGroupId\":\t1,\n\t\"EventType\":\t101,\n\t\"CallbackTs\":\t1
608086882372,\n\t\"EventInfo\":\t{\n\t\t\"RoomId\":\t20222,\n\t\t\"Event
Ts\":\t1608086882,\n\t\t\"UserId\":\t\"222222_phone\"\n\t}\n}";

$api = new  TlsEventSig($key, $data);
echo $api->genEventSig();
```

Golang

```
package main
import "fmt"
import (
    "crypto/hmac"
    "crypto/sha256"
    "encoding/base64"
)

func main () {
    var data = "
{\n\t\"EventGroupId\":\t1,\n\t\"EventType\":\t101,\n\t\"CallbackTs\":\t1
608086882372,\n\t\"EventInfo\":\t{\n\t\t\"RoomId\":\t20222,\n\t\t\"Event
Ts\":\t1608086882,\n\t\t\"UserId\":\t\"222222_phone\"\n\t}\n}"
    var key = "789"

    //JSRUN引擎2.0，支持多达30种语言在线运行，全仿真在线交互输入输出。
    fmt.Println(hmacsha256(data,key))
}

func hmacsha256(data string, key string) string {
    h := hmac.New(sha256.New, []byte(key))
    h.Write([]byte(data))
    return base64.StdEncoding.EncodeToString(h.Sum(nil))
}
```

自定义 TTS 协议

最近更新时间：2024-11-08 18:05:32

调用方式

POST：http://xxxxxxxxxxxx/api/v1/tts/stream

示例

HTTP 请求 Header

```
Content-Type: application/json;charset=UTF-8
X-Task-Id: task_id_value
X-Rquest-Id: request_id
X-Sdk-App-Id: SdkAppId
X-User-Id: UserId
X-Room-Id: RoomId
X-Room-Id-Type: "0"
Authorization: Bearer "API-KEY"
```

请求体示例

```
{
  "Text": "你好，世界！这是流式 TTS API 的测试。",
  "Format": "wav",
  "SampleRate": 16000,
  "Channel": 1
}
```

HTTP 请求 Header

字段	描述
Content-Type	application/json
charset	UTF-8
X-Task-Id	对话任务的 Id
X-Request-Id	此次请求的 Id，重试会携带相同的 RequestId
X-Sdk-App-Id	SDK的AppId

X-User-Id	用户Id
X-Room-Id	Romm Id
X-Room-Id-Type	Romm Id的类型，0 - 数字房间号，1 - 表示字符串房间号
Authorization	鉴权，格式为Bearer "API-KEY"

请求体

参数名称	必填	类型	描述
Text	是	String	语音文本
Format	否	String	期望输出的音频格式，如mp3， ogg_opus，pcm，wav，默认为wav，一期只支持pcm和wav
Sample Rate	否	Integer	音频采样率，默认为16000(16k)，推荐值为16000
Channel	否	Integer	音频通道，取值：1 或 2 默认 1

响应

需要根据 Content-Type 的值来确定合成是否成功。

- **如果成功，正常返回为二进制语音，不同的音频格式的 Content-Type 如下，需要在 HTTP 响应的 Header 设置 Transfer-Encoding: chunked。**

> ⚠ **注意：**
> Response Header 中必须包含 Content-Type 和 Transfer-Encoding，我们服务会强制检查。
> 比如：
> {
> **Transfer-Encoding: chunked;**
> **Content-Type: audio/L16**
> }

音频格式	Content-Type
mp3	audio/mpeg
ogg_opus	audio/ogg

pcm	audio/L16
wav	audio/wav

- **如果失败**，则会返回 json 结果，具体 header 信息为：Content-type: application/json。响应为：

```
{
  "error": {
    "code": "ERROR_CODE",
    "message": "A description of the error"
  }
}
```

行业实践教程
情感陪伴

最近更新时间：2024-09-02 17:35:11

场景描述

AI 情感陪伴产品迎来井喷式创新，并迅速走向多样化，包括角色扮演、情感陪聊、心理疗愈等多种类型。现有的 AI 语聊场景主要是基于 IM 场景的离线文字聊天或者语音聊天，随着GPT-4o的发布，把多模态大模型的应用场景提升到了实时的语音或者视频交互，用户得以沉浸在一个更为真实、互动的虚拟世界，享受前所未有的娱乐体验。
结合第三方的大模型和 TTS 等，腾讯云实时音视频（TRTC）让边消费边创作、剧情无限延伸且充满无限可能、实时互动与实时变化的情感陪伴体验成为现实，开发者可以轻松创建一个媲美GPT-4o所演示的AI实时交互体验。用户不仅能感受到高度个性化的陪伴，还能在互动中激发创作灵感，共同探索 AI 陪伴 + 实时互动中的无限可能。
腾讯云实时音视频（TRTC）技术的优势尤为显著。TRTC 提供了低延迟、高质量的语音和视频通信，确保了用户在与虚拟角色互动时的流畅性和真实感。此外，TRTC 的高并发处理能力使得大量用户能够同时在线，享受无缝的情感陪伴体验。

场景分类

场景分类	描述	核心需求
虚拟伴侣	用户可以通过定制“虚拟男友/女友”的形象、性格，使其成为一个更贴心的交流对象。这些AI恋人能够理解用户的情绪，提供安慰和支持，帮助用户建立和维持社交联系	• **个性化定制：**用户可以根据自己的喜好和需求定制 AI 恋人的性格、外貌、兴趣等，使得 AI 恋人更加贴近用户的期望 • **实时陪伴：**AI 恋人可以全天候在线，为用户提供即时的关心和支持，无论用户在何时何地遇到困难或需要倾诉 • **真实的交互体验：**理解语音中的情绪、在用户停顿时及时接过对话，音色按需选择
角色扮演	通过故事和剧情构建一个虚拟世界，用户可以选择不同的虚拟角色，在特定的剧情下与其进行文字或语音对话，本质是对 IP 内容的消费	• **沉浸式互动：**用户的核心诉求和爽点，在于和特定角色进行交互和扮演，建立与该角色的联结；或者通过和角色之间的剧情演绎，获得幻想和创作的快乐 • **音色记忆：**AI 角色需要能够记住并模仿特定人物的音色，这对于保持角色的一致性和真实感非常重要
偶像/网红 AI 分身	AI 分身通过高度还原偶像/网红的音色和语调，粉丝可以通过付费来获得与AI分身的专属互动，如情感陪伴、日常聊天等	• **专属音色和语料数据：**建立专属的语音和人格引擎，让用户感觉就像是沉浸式地与偶像本人在面对面交谈

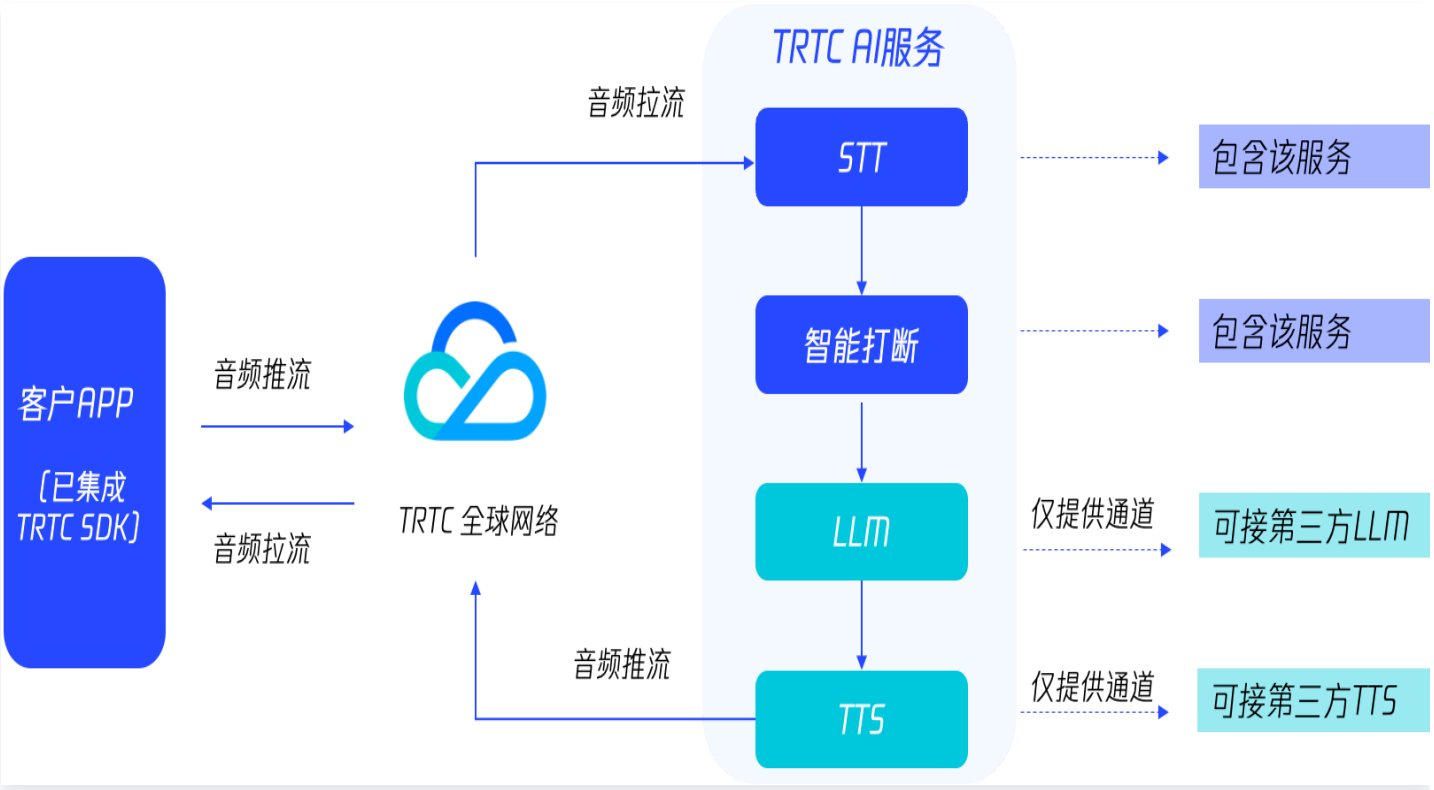
		• **多种互动方式变现：**通过 AI 分身，除了付费语音聊天外，网红可以开展多种形式的粉丝经济变现，如虚拟礼物、付费问答、专属活动等

场景需求说明

角色	核心场景	场景详情	核心需求
用户	和 AI 虚拟角色对话	通过语音和 AI 虚拟角色实时对话	• 超低延迟：期望与 AI 的交流能够毫无延迟，实现如同面对面交谈般的自然流畅体验，而非对讲机式的断断续续。 • 多音色：希望系统能够模拟或处理多种不同的声音，从而享受到更为丰富多彩的语音体验 • STT 准确：需要系统能够准确识别其语音，并将其转换成文本，以确保信息传递的准确性与高效性 • 随时打断：在任何时候都能够轻松地中断系统的输出或操作，以便更好地掌控对话节奏，提升交互效率 • 情感需求：AI 能够理解和响应用户的情感状态，能够提供更加人性化和富有同理心的交互体验
AI Bot	和用户进行对话	通过语音和用户实时对话	• 理解并响应用户的情感和需求 • 提供连贯、自然且富有同理心的对话 • 维护用户的隐私和安全 • 不断学习和适应用户的行为和偏好，以提供个性化的服务 • 在技术上保持先进性，以提供流畅和高质量的交互体验
开发者	开发并上线 AI 陪伴 App	搭建 AI 陪伴 App，快速上线，保障用户体验	• 确保用户在进行与 AI 的语音对话时能够体验到超低延迟的实时互动，保障对话的流畅性和自然性 • 支持微信小程序、Android、iOS、Web 等多平台，且能互通，降低多平台独立维护的成本 • 使用一站式服务，快速上线和迭代 AI 陪伴类产品，缩短产品上市时间 • 确保用户数据的安全性和隐私保护，遵守相关法律法规，建立用户信任

			• 扩展性灵活，可对接多家 LLM，为之后的 RAG 和 Fine-tune 提供基础

技术方案



核心功能

功能	AI情感陪聊场景应用
实时音频互动	用户可以通过类似“打电话”的方式，与 AI 虚拟角色进行一对一的语音交流。采用流式传输，这种实时互动不仅能够及时缓解用户的孤独感，还能提供即时的反馈和支持
STT	用户的语音会被精确识别，并实时转换成文字，发送给 LLM。LLM 会根据用户的语音内容和情感状态，提供相应的安慰和建议
LLM	支持接入第三方 LLM 或自有大模型，可结合 RAG（检索增强生成）/客户知识库，提供更加智能化和个性化的回复
智能打断	用户可以在任何时候打断 AI 的发言，表达自己的想法和感受。这种互动方式不仅增强了对话的自然性，还能让用户感受到更多的控制权和参与感。提供智能打断和手动打断两个模式，方便用户选择

TTS	仅提供通道，支持接入第三方 TTS，AI 虚拟角色可以以多种音色与用户交流，提供更加亲切和个性化的体验
AI 降噪	通过先进的AI降噪技术，确保对话在嘈杂环境中也能保持清晰，减少因噪声引起的语音中断或模糊，提升用户的沟通体验
弱网卡顿优化	即使在网络条件不佳的情况下，AI 实时对话也能保持流畅，确保用户在任何环境下都能获得稳定的情感支持
多平台互通	用户可以在微信小程序、iOS、Android、Web 等多个平台上无缝运行应用，随时随地发起和参与对话，极大提高了 AI 陪伴的便捷性和可达性

扩展功能

1. **在情感陪伴应用场景中添加文本、语音、图片的多模态交互**

   多模态交互能够提供更丰富、更自然的人机交互体验。通过结合文本、图像、音频等多种信息模态，AI 情感陪伴能够更好地理解用户的意图、情感和需求，从而提供更加个性化和适应性的响应。我们推荐用**即时通信 IM** 来实现该场景拓展，详情参见 即时通信 IM 产品介绍 。
2. **在情感陪伴应用中添加语聊房、多人群聊等玩法**

   越来越多的 AI 陪伴场景里面也集成了社交的功能，与朋友或其他用户一起讨论和分享，甚至分享和AI陪伴聊天的记忆。社交功能的集成使得AI陪伴场景更加生动有趣，增强了用户的使用黏性和满意度。我们推荐使用 语音聊天室 来实现该场景的拓展。

智能客服

最近更新时间：2024-09-03 18:25:12

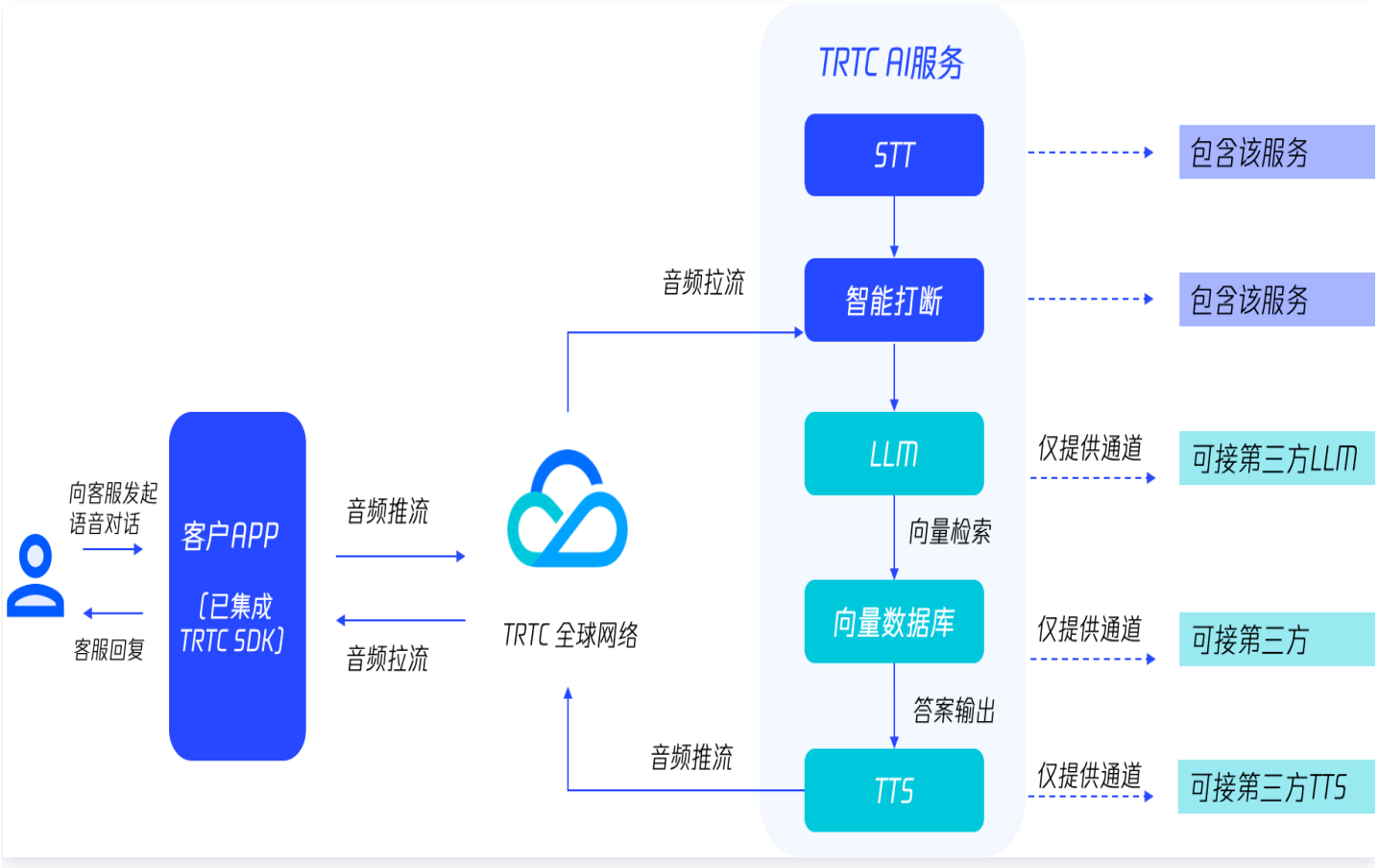
场景描述

智能语音客服是一种利用人工智能和语音识别技术来实现自动化交互和问题解决的客户服务系统。在 AI 大模型出现之前，智能客服主要利用自然语言处理和机器学习算法来理解客户的意图，依赖于预设的规则和知识库来进行问题解答。随着 LLM 的发展，越来越多的智能客服接入了大模型的能力，LLM 技术使智能语音客服能够更好地理解对话的上下文，从而实现连贯的对话交流。
TRTC 技术的引入，为智能语音客服带来了实时通信的能力。这意味着智能客服可以更加迅速地响应客户的需求，提供即时的反馈和解决方案。同时，TRTC 还支持多人通话、屏幕共享等功能，进一步提升了客户服务的效率和质量。

场景需求说明

场景分类	核心场景	场景详情	核心需求
用户	和智能客服对话	通过语音、文字、图片等方式和智能客服对话，解决特定问题	• 更自然的对话体验：与智能客服系统的对话更接近人与人之间的自然交流，能够实时对话 • 个性化服务：通过智能客服可以获得个性化的服务建议和解决方案
企业	接入智能客服	保障客户体验，让智能客服具备独立为用户解决问题的能力	• 客户体验：企业对客服团队的核心考核指标依然是客户满意度、响应速度和接通率。通过LLM+智能客服的方案，可进一步识别客户意图，并生成自然、流畅的回答，帮助智能客服提高对话质量和流畅度
AI客服	和用户对话	及时、准确地回答客户的问题，并能帮助客户解决问题	• 多平台互动，可以多平台接入，支持微信小程序、Android、iOS、Web 等多平台，且能互通，降低多平台独立维护的成本 • 多语言支持：特别是对于开展国际业务，结合大模型可以很好地对买家和卖家之间的对话语言进行转换，例如买家用英语发文，而卖家用中文回复，但双方看到的都是自己的偏好语言

技术方案



核心功能

功能	AI 智能客服场景应用
实时音频互动	流式传输技术可以确保语音和视频数据的连续性和稳定性，减少延迟和抖动，提供接近于真人客服通话的高质量体验。用户可以与智能客服系统进行更自然的对话，就像是在和真人客服交谈一样，这种互动体验可以显著提升用户满意度
STT	在智能客服场景中使用 STT，可以提高语音识别和情感识别的准确率，且可以利用历史对话记录和全局语境来提高语音识别的准确性和语义一致性
多语言支持	智能语音客服在全球范围内使用，需要适应多种语言和跨文化交流。精准的 STT 识别支持多种语言，包括英语、西班牙语、日语、韩语、中文以及23种方言和130种国际语言。
LLM	LLM 技术使智能语音客服能够更好地理解对话的上下文，从而实现连贯的对话交流。LLM 可以捕捉对话中的语义和语境信息，识别用户意图，并将上一轮对话的内容与当前对话关联起来
智能打断	用户可以在任何时候打断 AI 的发言，表达自己的想法和感受。这种互动方式不仅增强了对话的自然性，还能让用户感受到更多的控制权和参与感

TTS	仅提供通道，支持接入第三方 TTS，通过在模型中引入个性化的训练数据或调整模型的参数，可以生成符合特定要求的语音输出。智能语音客服可以根据用户的偏好或特定场景的需求，提供不同的语音风格
AI降噪	通过先进的 AI 降噪技术，确保对话在嘈杂环境中也能保持清晰，减少因噪声引起的语音中断或模糊，提升用户的沟通体验
弱网卡顿优化	即使在网络条件不佳的情况下，AI 实时对话也能保持流畅，确保用户在任何环境下都能获得稳定的客服支持
多平台互通	用户可以在微信小程序、iOS、Android、Web 等多个平台上无缝运行应用，随时随地发起和参与对话，极大提高了智能客服的便捷性和可达性

扩展功能

1. **在智能客服应用场景中添加文本、语音、图片的多模态交互**

   多模态交互能够提供更丰富、更自然的人机交互体验。通过结合文本、图像、音频等多种信息模态，AI 智能客服能够更好地理解用户的意图、情感和需求，从而提供更加个性化和适应性的响应。我们推荐用**即时通信 IM** 来实现该场景拓展，详情参见 即时通信 IM 产品 介绍。
2. **和通讯中心的互联互通**

   部分客服场景需要 RTC 和通话线路的平滑切换，从 RTC 线路切换到 SIP 线路，快速搭建集电话、在线交流、音视频通话为一体的客户联络平台。我们已有相应的解决方案，详情请见 云联络中心_TCCC 介绍。