麒麟 AI SDK 开发指南

1. 概述

1. 麒麟 AI SDK 将 AI 能力统一抽象封装为 C 语言接口，屏蔽了各个大模型的接口差异，降低了应用集成 AI 能力的门槛。麒麟 AI SDK 主要分为传统 AI 能力接口和生成式 AI 能力接口。传统 AI 能力接口包括文字识别、音频处理和向量化等能力，生成式 AI 能力接口包括文本生成和图像生成等能力接口；
2. 当前为1.1版本。

2. 前提条件

1. 基于已发布的银河麒麟操作系统（2503 版本）；
2. 端侧能力目前只能在 x86 和 arm 机器上运行，对于其他的机型效果无法保证，已适配的具体机型如下：

整机型号	具体配置
联想开天M90f G1s	CPU: D3000内存：16G独立显卡：Arise-GT10C0t硬盘：512G
联想开天M90h G1t	CPU: Hygon C86-3G (OPN:3350) 内存：16GB独立显卡：Arise-GT10C0t硬盘：512GB
联想开天 P90z G1t	CPU: ZHAOXIN KaiXian KX-7000 内存：16GB 独立显卡：Arise-GT10C0t 硬盘：1TB

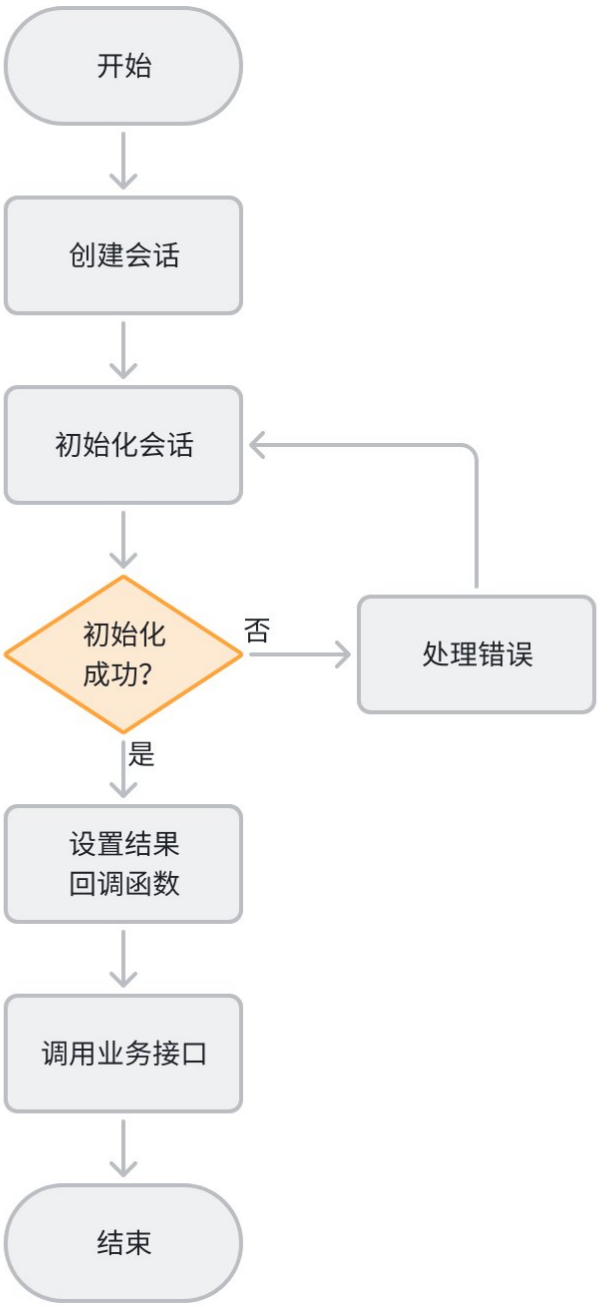
3. 在使用任何接口之前需要在“设置->AI 模块管理”中下载安装 AI 子系统。其次需要安装每种能力的开发包，具体参考每种能力的接口说明。

3. API 使用流程说明

3.1 使用流程说明

1. 使用任何接口之前需要创建会话实例；
2. 初始化会话实例，如果发生错误，初始化接口会返回具体的错误码；
3. 初始化成功之后设置结果回调函数，然后调用具体的业务接口；
4. 如果初始化失败并处理完错误，需要重新初始化会话实例；

5. 每个会话同时只能处理一个任务，如果同时调用多次业务接口，任务将串行执行，结果也是按照顺序返回；
6. 如果要执行并发，可创建多个会话进行实现。



3.2 模型配置

1. 模型配置是全局的，调用文本对话、文生图和语音相关接口之前，需要对模型配置进行相关的配置；
2. 模型配置可以在“设置->AI 模块管理”中进行配置。

4. API 详细说明

4.1 传统 AI 能力接口

4.1.1 文字识别

1. 文字识别接口将图像中的文字转换为文本，并提供文本行数和坐标等信息；
2. 目前仅支持使用 AI 子系统自带的端侧模型进行识别；
3. 通过“设置->AI 模块管理”下载 AI 子系统之后，不需要进行任何配置即可使用。

4.1.1.1 开发环境部署

```
sudo apt install libkysdk-coreai-vision-dev
```

4.1.1.2 文字识别会话

4.1.1.2.1 创建会话

头文件	<coreai/vision/textrecognition.h>
函数	TextRecognitionSession *text_recognition_create_session()
描述	创建文字识别会话
参数	无
返回值	TextRecognitionSession 类型的指针

4.1.1.2.2 初始化会话

头文件	<coreai/vision/textrecognition.h>
函数	int text_recognition_init_session(TextRecognitionSession *session)
描述	初始化文字识别会话
参数	• session：文字识别会话的指针
返回值	返回初始化的结果，初始化成功返回 0，否则返回对应的错误码

4.1.1.2.3 销毁会话

头文件	<coreai/vision/textrecognition.h>
函数	void text_recognition_destroy_session(TextRecognitionSession **session)

描述	销毁文字识别会话
参数	• session：文字识别会话的指针地址
返回值	无

4.1.1.2.4 设置文字识别结果回调函数

头文件	<coreai/vision/textrecognition.h>
函数	void text_recognition_result_set_callback(TextRecognitionSession *session, TextRecognitionResultCallback callback, void *user_data)
描述	设置文字识别结果的回调函数
参数	• session：文字识别会话的指针 • callback：TextRecognitionResultCallback 类型的结果回调函数 • user_data：调用者自定义的数据
返回值	无

4.1.1.2.5 设置模型配置信息

头文件	<coreai/vision/textrecognition.h>
函数	void text_recognition_set_model_config(TextRecognitionSession *session, TextRecognitionModelConfig *config)
描述	设置文字识别的模型配置信息
参数	• session：文字识别会话的指针 • config：模型配置
返回值	无

4.1.1.2.6 图片文件文字识别

头文件	<coreai/vision/textrecognition.h>
函数	void text_recognition_recognize_text_from_image_file_async(TextRecognitionSession *session, const char *image_file)
描述	进行图片文件的文字识别

参数	• session：文字识别会话的指针 • image_file：图片文件的路径
返回值	无

4.1.1.2.7 图片数据文字识别

头文件	<coreai/vision/textrecognition.h>
函数	void text_recognition_recognize_text_from_image_data_async(TextRecognitionSession *session, const char *image_data, unsigned int image_data_length)
描述	进行图片数据的文字识别
参数	• session：文字识别会话的指针 • image_data：待识别的图片数据指针，不需要转码，图片格式数据即可 • image_data_length：待识别的图片数据长度
返回值	无

4.1.1.3 设置配置信息

可以明确指定要使用的模型或者部署类型，当前版本可以无需关注，默认会使用 AI 子系统集成的模型。

4.1.1.3.1 创建模型配置

头文件	<coreai/vision/config.h>
函数	TextRecognitionModelConfig *text_recognition_model_config_create()
描述	创建模型配置实例
参数	无
返回值	模型配置实例指针

4.1.1.3.2 销毁模型配置示例

头文件	<coreai/vision/config.h>
函数	void text_recognition_model_config_destroy(TextRecognitionModelConfig **config)

描述	销毁模型配置实例
参数	• config：模型配置实例指针的地址
返回值	无

4.1.1.3.3 设置使用的模型名称

头文件	<coreai/vision/config.h>
函数	void text_recognition_model_config_set_name(TextRecognitionModelConfig *config, const char *name)
描述	设置要使用的模型名称，不指定时使用默认的模型
参数	• config：模型配置实例的指针 • name：设置的模型名字
返回值	无

4.1.1.3.4 设置使用的模型的部署类型

头文件	<coreai/vision/config.h>
函数	void text_recognition_model_config_set_deploy_type(TextRecognitionModelConfig *config, ModelDeployType type)
描述	设置使用的模型的部署类型，不指定时使用默认部署类型的模型
参数	• config：模型配置实例的指针 • type：指定的模型部署类型
返回值	无

4.1.1.4 结果处理

4.1.1.4.1 结果回调函数

头文件	<coreai/vision/textrecognition.h>
函数	typedef void (*TextRecognitionResultCallback)(TextRecognitionResult *result, void *user_data)
描述	进行图片数据的文字识别

参数	• result：TextRecognitionResult 类型的识别结果指针 • user_data：用户自定义的数据
返回值	无

4.1.1.4.2 结果解析

1. 获取一行文本中的内容

头文件	<coreai/vision/textrecognitionresult.h>
函数	const char *text_line_get_value(TextLine *text_line)
描述	获取一行文本中的内容
参数	• text_line：TextLine 类型的指针
返回值	返回该行文本的指针

2. 获取一行文本的角点位置信息（四个角的位置信息）

头文件	<coreai/vision/textrecognitionresult.h>
函数	PixelPoint *text_line_get_corner_points(TextLine *text_line, int *point_number)
描述	获取一行文本的角点位置信息（四个角的位置信息）
参数	• text_line：TextLine 类型的指针 • point_number：角点个数，固定输出为 4
返回值	返回四个点的坐标，顺序为左上、左下、右上、右下。

3. 获取识别结果的整体文本信息

头文件	<coreai/vision/textrecognitionresult.h>
函数	const char *text_recognition_result_get_value(TextRecognitionResult *result)
描述	获取识别结果的整体文本信息，不带格式
参数	• result：TextRecognitionResult 类型的识别结果指针
返回值	返回所有文本的内容

4. 获取识别的文本结果和行数

头文件	<coreai/vision/textrecognitionresult.h>
函数	TextLine **text_recognition_result_get_text_lines(TextRecognitionResult *result, int *line_count)
描述	获取识别的文本结果和行数
参数	• result：TextRecognitionResult 类型的识别结果 • line_count：文本行数
返回值	返回 TextLine*类型的数组的地址，TextLine 结果中包含文本信息和坐标信息

5. 获取错误码

头文件	<coreai/vision/textrecognitionresult.h>
函数	int text_recognition_result_get_error_code(TextRecognitionResult *result)
描述	获取识别结果中的错误码
参数	• result：TextRecognitionResult 类型的识别结果
返回值	返回具体的错误码，参考通用错误码和文字识别专有错误码

6. 获取错误信息

头文件	<coreai/vision/textrecognitionresult.h>
函数	const char *text_recognition_result_get_error_message(TextRecognitionResult *result)
描述	获取识别结果中的错误信息
参数	• result：TextRecognitionResult 类型的识别结果
返回值	如果发生错误则返回具体的错误信息，否则返回空

4.1.1.5 错误码

通用错误码可参考 4.3 章节，文字识别专有错误码如下：

头文件	<coreai/vision/error.h>

枚举	typedef enum { OCR_IMAGE_ERROR = 100, OCR_PARAM_INVALID, } OcrErrorCode;
描述	文字识别相关的错误码
成员	• OCR_IMAGE_ERROR：数据文件无效 • OCR_PARAM_INVALID：参数无效

4.1.1.6 示例

CMakeLists.txt

```
find_package(PkgConfig REQUIRED)
pkg_check_modules(GIO REQUIRED gio-unix-2.0)
include_directories(${GIO_INCLUDE_DIRS})

pkg_check_modules(KYSDK_AI_VISION kysdk-coreai-vision)
include_directories(${KYSDK_AI_VISION_INCLUDE_DIRS})
target_link_libraries(
    XXXX
    pthread
    ${GIO_LIBRARIES}
    ${KYSDK_AI_VISION_LIBRARIES}
)
```

从图像文件识别

代码块

```
#include <coreai/vision/textrecognition.h>
#include <gio/gio.h>
#include <gio/giotypes.h>

#include <iostream>
#include <thread>

const char *TEST_FILE_PATH = "/home/kylin/Kylinproject/test2.png";

void callback(TextRecognitionResult *result, void *user_data) {
    fprintf(stdout, "Start printing results.\n");

    int textLineNum = 0, pointsNum = 0;
```

```
    fprintf(stdout, "text        : %s\n",
            text_recognition_result_get_value(result));
    fprintf(stdout, "err code    : %i\n",
            text_recognition_result_get_error_code(result));
    fprintf(stdout, "err message : %s\n",
            text_recognition_result_get_error_message(result));

    _TextLine **textline =
        text_recognition_result_get_text_lines(result, &textLineNum);
    if (textline == nullptr) {
        fprintf(stderr, "The result is invalid, please check image\n");
        return;
    }

    for (int i = 0; i < textLineNum; ++i) {
        PixelPoint *point =
            text_line_get_corner_points(textline[i], &pointsNum);
        if (point == nullptr) {
            fprintf(stderr, "No point\n");
            return;
        }
        fprintf(stdout, "The %i line text: %s\n", i,
                text_line_get_value(textline[i]));

        for (int i = 0; i < pointsNum; i++) {
            printf("The corner points text %d: (%d, %d)\n", i, point[i].x,
                   point[i].y);
        }
    }

    if (user_data != nullptr) {
        const char *userData = static_cast<const char *>(user_data);
        fprintf(stdout, "%s\n", userData);
    } else {
        fprintf(stdout, "user data is nullptr\n");
    }

    fprintf(stdout, "Printing result completed.\n");
}

void test01_OcrFromFile() {
    const char *userData = "Test genai vision from image file\n";

    TextRecognitionSession *session = text_recognition_create_session();

    TextRecognitionModelConfig *config =
text_recognition_model_config_create();
```

```
    text_recognition_model_config_set_name(config, "vision");
    text_recognition_model_config_set_deploy_type(config,
                                                  ModelDeployType::OnDevice);

    text_recognition_set_model_config(session, config);

    text_recognition_init_session(session);

    text_recognition_result_set_callback(session, callback, (void *)userData);

    text_recognition_recognize_text_from_image_file_async(session,
                                                          TEST_FILE_PATH);

    GMainLoop *pMainLoop = g_main_loop_new(nullptr, false);

    std::thread ctrlThread([&session, &config, pMainLoop] {
        while (std::getchar() != '\n') {
        }
        text_recognition_destroy_session(&session);
        text_recognition_model_config_destroy(&config);
        g_main_loop_quit(pMainLoop);
    });

    ctrlThread.detach();
    g_main_loop_run(pMainLoop);
    g_main_loop_unref(pMainLoop);
}
```

从图像数据识别

```
#include <coreai/vision/textrecognition.h>
#include <gio/gio.h>
#include <gio/giotypes.h>

#include <filesystem>
#include <fstream>
#include <iostream>
#include <thread>
#include <vector>

const char *TEST_DATA_PATH = "/home/kylin/Kylinproject/test1";

void callback(TextRecognitionResult *result, void *user_data) {
    fprintf(stdout, "Start printing results.\n");

```

```
    int textLineNum = 0, pointsNum = 0;
    fprintf(stdout, "text        : %s\n",
            text_recognition_result_get_value(result));
    fprintf(stdout, "err code    : %i\n",
            text_recognition_result_get_error_code(result));
    fprintf(stdout, "err message : %s\n",
            text_recognition_result_get_error_message(result));

    _TextLine **textline =
        text_recognition_result_get_text_lines(result, &textLineNum);
    if (textline == nullptr) {
        fprintf(stderr, "The result is invalid, please check image\n");
        return;
    }

    for (int i = 0; i < textLineNum; ++i) {
        PixelPoint *point =
            text_line_get_corner_points(textline[i], &pointsNum);
        if (point == nullptr) {
            fprintf(stderr, "No point\n");
            return;
        }
        fprintf(stdout, "The %i line text: %s\n", i,
                text_line_get_value(textline[i]));

        for (int i = 0; i < pointsNum; i++) {
            printf("The corner points text %d: (%d, %d)\n", i, point[i].x,
                   point[i].y);
        }
    }

    if (user_data != nullptr) {
        const char *userData = static_cast<const char *>(user_data);
        fprintf(stdout, "%s\n", userData);
    } else {
        fprintf(stdout, "user data is nullptr\n");
    }

    fprintf(stdout, "Printing result completed.\n");
}

std::vector<char> readImageData(const std::string &filePath) {
    std::ifstream file(filePath, std::ios::binary);
    if (!file.is_open()) {
        fprintf(stderr, "Failed to open file: %s\n", filePath.c_str());
        return {};
    }
```

```
63
64      file.seekg(0, std::ios::end);
65      std::streampos fileSize = file.tellg();
66      file.seekg(0, std::ios::beg);
67      std::vector<char> imageData(fileSize);
68      file.read(reinterpret_cast<char *>(imageData.data()), fileSize);
69      return imageData;
70  }
71
72  void test02_OcrFromData() {
73      namespace fs = std::filesystem;
74      if (not fs::exists(TEST_DATA_PATH)) {
75          fprintf(stderr, "error\n");
76          return;
77      }
78
79      const char *userData = "Test genai vision from image data\n";
80      const std::vector<char> imageData = readImageData(TEST_DATA_PATH);
81
82      TextRecognitionSession *session = text_recognition_create_session();
83
84      TextRecognitionModelConfig *config =
    text_recognition_model_config_create();
85
86      text_recognition_set_model_config(session, config);
87
88      text_recognition_init_session(session);
89
90      text_recognition_result_set_callback(session, callback, (void*)userData);
91
92      text_recognition_recognize_text_from_image_data_async(
93          session, imageData.data(), imageData.size());
94
95      GMainLoop *pMainLoop = g_main_loop_new(nullptr, false);
96
97      std::thread ctrlThread([&session, &config, pMainLoop] {
98          while (std::getchar() != '\n') {
99          }
100         text_recognition_model_config_destroy(&config);
101         text_recognition_destroy_session(&session);
102         g_main_loop_quit(pMainLoop);
103     });
104     ctrlThread.detach();
105     g_main_loop_run(pMainLoop);
106     g_main_loop_unref(pMainLoop);
107 }
108
```

4.1.2 音频处理

4.1.2.1 开发环境部署

```
sudo apt install libkysdk-coreai-speech-dev
```

4.1.2.2 语音识别

1. 将语音识别接口将音频信息转换为文本；
2. 目前仅支持中文；
3. 支持流式和非流式语音识别；
4. 支持识别发言人（如果云端服务支持的话）；
5. 目前仅支持云端服务的形式，端侧模型咱不支持；
6. 需要在“设置->AI 模块管理”中进行配置才能使用。

4.1.2.2.1 创建会话

头文件	<coreai/speech/recognizer.h>
函数	SpeechRecognitionSession *speech_recognizer_create_session()
描述	创建语音识别会话
参数	无
返回值	SpeechRecognitionSession 类型的指针

4.1.2.2.2 初始化会话

头文件	<coreai/speech/recognizer.h>
函数	int speech_recognizer_init_session(SpeechRecognitionSession *session)
描述	初始化语音识别会话
参数	• session：语音识别会话的指针
返回值	返回初始化的结果，初始化成功返回 0，否则返回对应的错误码

4.1.2.2.3 销毁会话

头文件	<coreai/speech/recognizer.h>

函数	void speech_recognizer_destroy_session(SpeechRecognitionSession **session)
描述	销毁语音识别会话
参数	• session：语音识别会话指针的地址
返回值	无

4.1.2.2.4 设置语音识别结果回调函数

头文件	<coreai/speech/recognizer.h>
函数	void speech_recognizer_result_set_callback(SpeechRecognitionSession *session, SpeechRecognitionResultCallback callback, void *user_data)
描述	设置语音识别的结果回调函数
参数	• session：语音识别会话的指针 • callback: SpeechRecognitionResultCallback 类型的结果回调函数 • user_data: 调用者自定义的数据
返回值	无

4.1.2.2.5 设置输入音频配置信息

头文件	<coreai/speech/recognizer.h>
函数	void speech_recognizer_set_audio_config(SpeechRecognitionSession *session, AudioConfig *audio_config)
描述	设置音频相关的配置，比如输入音频的来源等
参数	• session：语音识别会话的指针 • audio_config: 语音相关的配置
返回值	无

4.1.2.2.6 设置模型配置信息

头文件	<coreai/speech/recognizer.h>
函数	void speech_recognizer_set_model_config(SpeechRecognitionSession *session, SpeechModelConfig *config)

描述	设置模型配置信息
参数	• session：文字识别会话的指针 • config：模型配置
返回值	无

4.1.2.2.7 开始异步流式语音识别

头文件	<coreai/speech/recognizer.h>
函数	void speech_recognizer_start_continuous_recognition_async(SpeechRecognitionSession *session)
描述	开始流式异步语音识别，如果使用的音频流，建议每 40ms 发送 1280 个字节。结果通过 callback 异步返回。
参数	• session：语音识别会话的指针
返回值	无

4.1.2.2.8 停止异步流式语音识别

头文件	<coreai/speech/recognizer.h>
函数	void speech_recognizer_stop_continuous_recognition_async(SpeechRecognitionSession *session)
描述	停止异步流式语音识别
参数	• session：语音识别会话的指针
返回值	无

4.1.2.2.9 进行一次性语音识别

头文件	<coreai/speech/recognizer.h>
函数	void speech_recognizer_recognize_once_async(SpeechRecognitionSession *session)
描述	进行一次性语音识别，识别完整个文件或者整段数据时返回结果。结果通过 callback 异步返回。
参数	session：语音识别会话的指针

返回值	无

4.1.2.2.10 结果处理

4.1.2.2.10.1 结果回调函数

头文件	<coreai/speech/result.h>
类型	typedef void (*SpeechRecognitionResultCallback)(SpeechRecognitionResult *result, void *user_data)
描述	语音识别结果回调函数类型
参数	• result：SpeechRecognitionResult 类型的指针 • user_data: 用户数据
返回值	无

4.1.2.2.10.2 结果解析

1. 获取识别结果状态

头文件	<coreai/speech/result.h>
函数	SpeechResultReason speech_recognition_result_get_reason(SpeechRecognitionResult *result);
描述	获取语音识别结果的状态
参数	语音识别结果的指针
返回值	语音识别结果的状态

2. 获取识别的文本数据

头文件	<coreai/speech/result.h>
函数	const char *speech_recognition_result_get_text(SpeechRecognitionResult *result)
描述	获取语音识别结果中的文本数据
参数	语音识别结果的指针

返回值	语音识别结果的文本数据

3. 获取发言人的 id

头文件	<coreai/speech/result.h>
函数	int speech_recognition_result_get_speaker_id(SpeechRecognitionResult *result)
描述	获取语音识别结果中的说话人 id
参数	语音识别结果的指针
返回值	• 如果识别到发言人，返回大于 0 的 id • 否则返回 -1

4. 获取错误码

头文件	<coreai/speech/result.h>
函数	int speech_recognition_result_get_error_code(SpeechRecognitionResult *result)
描述	获取语音识别结果中的错误码
参数	语音识别结果的指针
返回值	具体的错误码

5. 获取具体错误信息

头文件	<coreai/speech/result.h>
函数	const char *speech_recognition_result_get_error_message(SpeechRecognitionResult *result);
描述	获取语音识别结果中的具体错误信息
参数	语音识别结果的指针
返回值	具体的错误信息

4.1.2.3 语音合成

1. 将纯文本内容合成为音频；
2. 暂时不支持设置发音人；
3. 暂时仅支持中文；
4. 目前仅支持云端服务的形式；
5. 需要在“设置->AI 模块管理”中进行配置之后才能使用。

4.1.2.3.1 创建会话

头文件	<coreai/speech/synthesizer.h>
函数	SpeechSynthesizerSession *speech_synthesizer_create_session()
描述	创建语音合成的会话
参数	无
返回值	语音合成会话的指针

4.1.2.3.2 初始化会话

头文件	<coreai/speech/synthesizer.h>
函数	int speech_synthesizer_init_session(SpeechSynthesizerSession *session)
描述	初始化语音合成的会话
参数	语音合成会话的指针
返回值	初始化结果，0 表示成功；大于 0 时表示具体的错误码

4.1.2.3.3 销毁会话

头文件	<coreai/speech/synthesizer.h>
函数	void speech_synthesizer_destroy_session(SpeechSynthesizerSession **session)
描述	销毁语音合成会话
参数	语音合成会话指针的地址

返回值	无

4.1.2.3.4 设置语音合成结果回调函数

头文件	<coreai/speech/synthesizer.h>
函数	void speech_synthesizer_result_set_callback(SpeechSynthesizerSession *session, SpeechSynthesisResultCallback callback, void *user_data)
描述	设置语音合成的结果回调函数
参数	• session: 语音合成会话的指针 • callback: 语音合成的结果回调函数
返回值	无

4.1.2.3.5 设置输出音频配置信息

头文件	<coreai/speech/synthesizer.h>
函数	void speech_synthesizer_set_audio_config(SpeechSynthesizerSession *session, AudioConfig *audio_config)
描述	设置语音合成输出音频的相关配置
参数	• session: 语音合成会话的指针 • audio_config: 具体的音频配置
返回值	无

4.1.2.3.6 设置模型配置信息

头文件	<coreai/speech/synthesizer.h>
函数	void speech_synthesizer_set_model_config(SpeechRecognitionSession *session, SpeechModelConfig *config)
描述	设置模型配置信息
参数	• session：文字识别会话的指针 • config：模型配置
返回值	无

4.1.2.3.7 进行语音合成

头文件	<coreai/speech/synthesizer.h>
函数	void speech_synthesizer_synthesize_text_async(SpeechSynthesizerSession *session, const char *text, uint32_t text_length)
描述	将文本内容合成为语音数据
参数	• session：语音合成会话的指针 • text: 文本数据指针 • text_length: 文本长度
返回值	无

4.1.2.3.8 停止播放音频

头文件	<coreai/speech/synthesizer.h>
函数	void speech_synthesizer_stop_speaking(SpeechSynthesizerSession *session)
描述	停止语音播放，当音频输出配置为系统播放器时该接口生效
参数	session：语音合成会话的指针
返回值	无

4.1.2.3.9 结果处理

4.1.2.3.9.1 结果回调函数

头文件	<coreai/speech/result.h>
类型	typedef void (*SpeechSynthesisResultCallback)(SpeechSynthesisResult *result, void *user_data)
描述	语音合成结果回调函数类型
参数	• result：SpeechSynthesisResult 类型的指针 • user_data: 用户数据
返回值	无

4.1.2.3.9.2 结果解析

1. 获取语音合成结果的状态

头文件	<coreai/speech/result.h>
函数	SpeechResultReason speech_synthesis_result_get_reason(SpeechSynthesisResult *result)
描述	获取语音合成结果的状态
参数	语音合成结果的指针
返回值	语音合成结果的状态

2. 获取语音合成结果的数据

头文件	<coreai/speech/result.h>
函数	const uint8_t *speech_synthesis_result_get_data(SpeechSynthesisResult *result, uint8_t *data_length)
描述	获取语音合成的数据
参数	• result: 语音合成结果的指针 • data_length: 输出参数，音频数据的长度
返回值	语音合成的音频数据的指针

3. 获取语音合成结果的音频数据格式

头文件	<coreai/speech/result.h>
函数	int speech_synthesis_result_get_audio_format(SpeechSynthesisResult *result)
描述	获取语音合成结果的音频数据格式
参数	语音合成结果的指针
返回值	具体的音频数据格式

4. 获取语音合成结果的音频数据采样率

头文件	<coreai/speech/result.h>

函数	int speech_synthesis_result_get_audio_rate(SpeechSynthesisResult *result)
描述	获取语音合成结果的音频数据采样率
参数	语音合成结果的指针
返回值	具体的音频数据采样率

5. 获取语音合成结果的音频数据通道数

头文件	<coreai/speech/result.h>
函数	int speech_synthesis_result_get_audio_channel(SpeechSynthesisResult *result)
描述	获取语音合成结果的音频数据通道数
参数	语音合成结果的指针
返回值	具体的音频数据通道数

6. 获取语音合成结果的错误码

头文件	<coreai/speech/result.h>
函数	int speech_synthesis_result_get_error_code(SpeechSynthesisResult *result)
描述	获取语音合成结果的错误码
参数	语音合成结果的指针
返回值	具体的错误码

7. 获取语音合成结果的错误信息

头文件	<coreai/speech/result.h>
函数	const char *speech_synthesis_result_get_error_message(SpeechSynthesisResult *result)
描述	获取语音合成结果的具体错误信息
参数	语音合成结果的指针
返回值	具体的错误信息

4.1.2.4 音频结果状态

头文件	<coreai/speech/result.h>
枚举	typedef enum { SPEECH_ERROR_OCCURRED = 1， SPEECH_RECOGNITION_STARTED = 2， SPEECH_RECOGNIZING = 3， SPEECH_RECOGNIZED = 4， SPEECH_RECOGNITION_COMPLETED = 5， SPEECH_SYNTHESIS_STARTED = 6， SPEECH_SYNTHESIZING = 7， SPEECH_SYNTHESIS_COMPLETED = 8 } SpeechResultReason;
描述	• SPEECH_ERROR_OCCURRED：语音识别或者合成过程中出错 • SPEECH_RECOGNITION_STARTED：语音识别已启动 • SPEECH_RECOGNIZING：正在进行语音识别，中间结果 • SPEECH_RECOGNIZED：语音识别的最终结果，在SPEECH_RECOGNIZING的基础上经过修正的结果 • SPEECH_RECOGNITION_COMPLETED：语音识别完成 • SPEECH_SYNTHESIS_STARTED：语音合成已启动 • SPEECH_SYNTHESIZING：正在进行语音合成 • SPEECH_SYNTHESIS_COMPLETED：语音合成已完成

4.1.2.5 音频配置

4.1.2.5.1 输入音频配置 - 语音识别

1. 配置输入音频数据系统默认麦克风获取，适用于流式语音识别

头文件	<coreai/speech/audioconfig.h>
函数	AudioConfig *audio_config_create_continuous_audio_input_from_default_microphone()
描述	创建音频配置，输入音频数据从默认麦克风中获取
参数	无

返回值	音频配置实例指针

2. 配置输入音频数据从数据流中获取，适用于流式语音识别

头文件	<coreai/speech/audioconfig.h>
函数	AudioConfig *audio_config_create_continuous_audio_input_from_audio_data_stream(AudioDataStream *stream)
描述	创建音频配置，使用音频数据流作为输入音频
参数	stream: 音频数据流
返回值	音频配置实例指针

3. 配置输入音频从 pcm 数据中获取，适用于一次性语音识别

头文件	<coreai/speech/audioconfig.h>
函数	AudioConfig *audio_config_create_once_audio_input_from_pcm_data(const uint8_t *audio_data, uint32_t data_length)
描述	创建音频配置，使用 pcm 音频数据作为输入音频
参数	• audio_data: pcm 音频数据指针 • data_length: pcm 音频数据长度
返回值	音频配置指针

4. 配置输入音频从 pcm 文件中获取数据，适用于一次性语音识别

头文件	<coreai/speech/audioconfig.h>
函数	AudioConfig *audio_config_create_once_audio_input_from_pcm_file(const char *pcm_file)
描述	创建音频配置，使用 pcm 文件作为输入音频
参数	pcm_file: pcm 文件
返回值	音频配置指针

4.1.2.5.2 输出音频配置 - 语音合成

1. 配置语音以原始数据输出

头文件	<coreai/speech/audioconfig.h>
函数	AudioConfig *audio_config_create_audio_output_from_pcm_data()
描述	创建音频配置，将合成的音频以原始数据形式输出。结果通过 callback 异步返回。
参数	无
返回值	音频配置实例指针

2. 配置语音输出到 pcm 文件

头文件	<coreai/speech/audioconfig.h>
函数	AudioConfig *audio_config_create_audio_output_from_pcm_file_name(const char *pcm_file)
描述	创建音频配置，将合成的音频输出到 pcm 文件
参数	pcm_file: 输出保存的 pcm 文件
返回值	音频配置实例指针

3. 配置语音输出到系统默认扬声器

头文件	<coreai/speech/audioconfig.h>
函数	AudioConfig *audio_config_create_audio_output_from_default_speaker()
描述	创建音频配置，使用系统默认的扬声器作为音频输出
参数	无
返回值	音频配置实例指针

4.1.2.6 模型配置信息

4.1.2.6.1 创建模型配置

头文件	<coreai/speech/config.h>
函数	SpeechModelConfig *speech_model_config_create()
描述	创建模型配置实例

参数	无
返回值	模型配置实例指针

4.1.2.6.2 销毁模型配置实例

头文件	<coreai/speech/config.h>
函数	void speech_model_config_destroy(SpeechModelConfig **config)
描述	销毁模型配置实例
参数	• config：模型配置的二级指针
返回值	无

4.1.2.6.3 设置使用的模型名称

头文件	<coreai/speech/config.h>
函数	void speech_model_config_set_name(SpeechModelConfig *config, const char *name)
描述	设置模型名称
参数	• config：模型配置的实例指针 • name：设置的模型名字
返回值	无

4.1.2.6.4 设置使用的模型类型

头文件	<coreai/speech/config.h>
函数	void speech_model_config_set_deploy_type(SpeechModelConfig *config, ModelDeployType type)
描述	设置模型类型
参数	• config：模型配置的实例指针 • ModelDeployType：设置的模型类型
返回值	无

4.1.2.7 错误码

通用错误码参考 4.3 节，语音处理专有错误码如下：

头文件	<coreai/speech/error.h>
枚举	typedef enum { SPEECH_RECOGNITION_AUDIO_DATA_SIZE_INVALID = 100, SPEECH_SYNTHESIS_TEXT_LENGTH_INVALID, SPEECH_PARAM_INVALID, SPEECH_DEFAULT_MICROPHONE_INVALID, SPEECH_UNKNOWN_ERROR, SPEECH_UNSUPPORTED_LANGUAGE } SpeechErrorCode;
描述	语音相关的错误码
成员	• SPEECH_RECOGNITION_AUDIO_DATA_SIZE_INVALID：音频大小超限 • SPEECH_SYNTHESIS_TEXT_LENGTH_INVALID：输入文本长度超限 • SPEECH_PARAM_INVALID： 配置参数不合法 • SPEECH_DEFAULT_MICROPHONE_INVALID：未配置默认麦克风 • SPEECH_UNKNOWN_ERROR：未知错误 • SPEECH_UNSUPPORTED_LANGUAGE：不支持的语言 • 其余通用错误码，具体参考 3.3 节

4.1.2.8 示例

4.1.2.8.1 前提条件

在“设置->AI 模块管理”中已经对语音相关的模型进行了配置。

4.1.2.8.2 语音识别

配置 CMakeLists.txt

```
find_package(PkgConfig REQUIRED)
pkg_check_modules(GIO REQUIRED gio-unix-2.0)
include_directories(${GIO_INCLUDE_DIRS})

pkg_check_modules(KYAISPEECH kysdk-coreai-speech)
include_directories(${KYAISPEECH_INCLUDE_DIRS})
target_link_libraries(
    xxx
```

```
    pthread
    ${GIO_LIBRARIES}
    ${KYAISPEECH_LIBRARIES}
)
```

具体 demo：

```
#include <filesystem>
#include <fstream>
#include <vector>

// glib header
#include <gio/gio.h>

// kysdk-ai header
#include <coreai/speech/recognizer.h>

const char *PCM_FILE_PATH = "xxx.pcm";

std::vector<uint8_t> readAudioData(const std::string &filePath) {
    std::ifstream file(filePath, std::ios::binary);
    if (!file.is_open()) {
        return {};
    }

    file.seekg(0, std::ios::end);
    std::streampos fileSize = file.tellg();
    file.seekg(0, std::ios::beg);
    std::vector<uint8_t> audioData(fileSize);
    file.read(reinterpret_cast<char *>(audioData.data()), fileSize);
    return audioData;
}

void callback(SpeechRecognitionResult *result, void *user_data) {
    fprintf(stdout, "Start printing speech recognition results.\n");
    fprintf(stdout, "Speech recognition errorcode: %d\n",
            speech_recognition_result_get_error_code(result));
    fprintf(stdout, "Speech recognition error message: %s\n",
            speech_recognition_result_get_error_message(result));

    int resultType = speech_recognition_result_get_reason(result);
    const char *resultData = speech_recognition_result_get_text(result);
    fprintf(stdout, "Speech recognition result: %s\n", resultData);
    int resultErrorCode = speech_recognition_result_get_error_code(result);

```

```
    fprintf(stdout, "Printing speech recognition result completed.\n");
}

void Test_RecognitionOnce() {
    GMainLoop *pMainLoop = g_main_loop_new(nullptr, false);

    if (not std::filesystem::exists(PCM_FILE_PATH)) {
        fprintf(stderr, "Flie not exists !\n");
        return;
    }
    std::vector<uint8_t> audioData = readAudioData(PCM_FILE_PATH);

    SpeechRecognitionSession *session = speech_recognizer_create_session();

    SpeechModelConfig *modelconfig = speech_model_config_create();
    speech_model_config_set_name(modelconfig,
                                 "讯飞-语音大模型");  //或"百度-语音大模型"
    speech_model_config_set_deploy_type(modelconfig,
                                        ModelDeployType::PublicCloud);
    speech_recognizer_set_model_config(session, modelconfig);

    speech_recognizer_init_session(session);

    speech_recognizer_result_set_callback(session, callback, nullptr);

    auto *config = audio_config_create_once_audio_input_from_pcm_data(
        audioData.data(), audioData.size());

    speech_recognizer_set_audio_config(session, config);

    speech_recognizer_recognize_once_async(session);

    g_main_loop_run(pMainLoop);
    g_main_loop_unref(pMainLoop);
}
```

4.1.2.8.3 语音合成

```
#include <fstream>
#include <filesystem>
#include <vector>
#include <coreai/speech/synthesizer.h>
#include <gio/gio.h>

static void writeBinaryDataToFile(const std::string &filename,
```

```
                                    const std::vector<char> &data) {
    if (data.size() == 0) {
        fprintf(stderr, "Data is empty!\n");
        return;
    }
    std::ofstream outputFile(filename, std::ios::out | std::ios::binary |
std::ios::app);
    if (!outputFile.is_open()) {
        fprintf(stderr, "File open failed!\n");
        return;
    }
    outputFile.write(reinterpret_cast<const char*>(data.data()), data.size());
    outputFile.close();
}

void onSynthesisResult(SpeechSynthesisResult *result, void *userData) {
    fprintf(stdout, "Start writing the synthesized results to a file.\n");

    const char *userdata = static_cast<const char*>(userData);

    uint32_t audioDataLength;
    const uint8_t* audioData = speech_synthesis_result_get_data(result,
&audioDataLength);
    SpeechResultReason resultType =
speech_synthesis_result_get_reason(result);

    std::fprintf(
        stdout,
        "test Synthesis result reason=%i audioDataLength=%i userData=%s \n",
        (int)resultType, (int)audioDataLength, userdata);

    std::vector<char> data {audioData, audioData + audioDataLength };
    writeBinaryDataToFile("../testsynthesis.pcm", data);

    fprintf(stdout, "Write completed.\n");
}

void testSynthesisOutputPcmData() {
    GMainLoop *pMainLoop = g_main_loop_new(nullptr, false);

    auto *synthesizerConfig =
audio_config_create_audio_output_from_pcm_data();

    SpeechSynthesizerSession *synSession =
speech_synthesizer_create_session();

    SpeechModelConfig *modelconfig = speech_model_config_create();
```

```
50      speech_model_config_set_name(modelconfig, "讯飞-语音大模型");//或"百度-语音大
    模型"
51      speech_model_config_set_deploy_type(modelconfig,
52                                          ModelDeployType::PublicCloud);
53      speech_synthesizer_set_model_config(synSession, modelconfig);
54
55      speech_synthesizer_result_set_callback(synSession, onSynthesisResult,
    nullptr);
56      speech_synthesizer_init_session(synSession);
57
58      audio_config_set_input_audio_rate(synthesizerConfig, 16000);
59      speech_synthesizer_set_audio_config(synSession, synthesizerConfig);
60
61      speech_synthesizer_synthesize_text_async(synSession, "你好", 100);
62
63      int stopErrorCode = speech_synthesizer_stop_speaking(synSession);
64
65      g_main_loop_run(pMainLoop);
66      g_main_loop_unref(pMainLoop);
67  }
```

4.1.3 向量化

1. 将文本、图片（非结构化数据）转换为数值向量。

4.1.3.1 开发环境部署

`sudo apt install libkylin-coreai-embedding-dev`

4.1.3.2 文本向量化

4.1.3.2.1 创建会话

头文件	<coreai/embedding/embedding.h>
函数	TextEmbeddingSession *text_embedding_create_session();
描述	创建文本向量化会话
参数	无
返回值	文本向量化会话指针

4.1.3.2.2 初始化会话

头文件	<coreai/embedding/embedding.h>
函数	int text_embedding_init_session(TextEmbeddingSession *session);
描述	初始化会话
参数	session: 文本向量化会话指针
返回值	成功时返回 0，否则返回具体的错误码

4.1.3.2.3 销毁对话

头文件	<coreai/embedding/embedding.h>
函数	void text_embedding_destroy_session(TextEmbeddingSession **session);
描述	销毁会话
参数	session: 文本向量化会话指针的地址
返回值	无

4.1.3.2.4 获取文本向量化模型信息

头文件	<coreai/embedding/embedding.h>
函数	bool text_embedding_get_model_info(TextEmbeddingSession *session, char **model_info);
描述	获取文本向量化模型信息，需要调用embedding_model_info_destroy销毁资源
参数	• session: 文本向量化会话指针 • model_info: 模型信息指针的地址
返回值	true:成功，false:失败

4.1.3.2.5 向量化文本（同步）

头文件	<coreai/embedding/embedding.h>
函数	bool text_embedding(TextEmbeddingSession *session, const char *text, EmbeddingResult **result);
描述	向量化文本同步接口，需要调用embedding_result_destroy销毁资源

参数	• session: 文本向量化会话指针 • text: 文本 • result: EmbeddingResult类型指针的地址
返回值	true:成功，false:失败

4.1.3.2.6 向量化文本（异步）

头文件	<coreai/embedding/embedding.h>
函数	void text_embedding_async(TextEmbeddingSession *session, const char *text, TextEmbeddingResultCallback callback, void *callback_user_data);
描述	向量化文本异步接口
参数	• session: 文本向量化会话指针 • text: 文本 • callback: 结果回调函数 • callback_user_data: 用户数据
返回值	无

4.1.3.3 图像向量化

4.1.3.3.1 创建会话

头文件	<coreai/embedding/embedding.h>
函数	ImageEmbeddingSession *image_embedding_create_session();
描述	创建图像向量化会话
参数	无
返回值	图像向量化会话指针

4.1.3.3.2 初始化会话

头文件	<coreai/embedding/embedding.h>
函数	int image_embedding_init_session(ImageEmbeddingSession *session);
描述	初始化会话

参数	session: 图像向量化会话指针
返回值	成功时返回 0，否则返回具体的错误码

4.1.3.3.3 销毁对话

头文件	<coreai/embedding/embedding.h>
函数	void image_embedding_destroy_session(ImageEmbeddingSession **session);
描述	销毁会话
参数	session: 图像向量化会话指针的地址
返回值	无

4.1.3.3.4 获取图像向量化模型信息

头文件	<coreai/embedding/embedding.h>
函数	bool image_embedding_get_model_info(ImageEmbeddingSession *session, char **model_info);
描述	获取图像向量化模型信息，需要调用embedding_model_info_destroy销毁资源
参数	• session: 图像向量化会话指针 • model_info: 模型信息指针的地址
返回值	true:成功，false:失败

4.1.3.3.5 图像向量化模型向量化文本（同步）

头文件	<coreai/embedding/embedding.h>
函数	bool text_embedding_by_image_model(ImageEmbeddingSession *session,const char *text, EmbeddingResult **result);
描述	通过同步的方式图像向量化模型向量化文本，需要调用embedding_result_destroy销毁资源
参数	• session: 图像向量化会话指针 • text: 文本 • result: EmbeddingResult类型指针的地址

返回值	true:成功，false:失败

4.1.3.3.6 向量化图片（同步）

4.1.3.3.6.1 通过传入图片文件路径的方式

头文件	<coreai/embedding/embedding.h>
函数	bool image_embedding_by_image_file(ImageEmbeddingSession *session, const char *image_file, EmbeddingResult **result);
描述	通过同步的方式向量化图片，需要调用embedding_result_destroy销毁资源
参数	• session: 图像向量化会话指针 • image_file: 图片路径 • result: EmbeddingResult类型指针的地址
返回值	true:成功，false:失败

4.1.3.3.6.2 通过传入base64图片数据的方式

头文件	<coreai/embedding/embedding.h>
函数	bool image_embedding_by_base64_image_data(ImageEmbeddingSession *session, const unsigned char *image_data, unsigned int image_data_length, EmbeddingResult **result);
描述	通过同步的方式向量化图片，需要调用embedding_result_destroy销毁资源
参数	• session: 图像向量化会话指针 • image_data: base64图片数据 • image_data_length: base64图片数据的长度 • result: EmbeddingResult类型指针的地址
返回值	true:成功，false:失败

4.1.3.3.7 图像向量化模型向量化文本（异步）

头文件	<coreai/embedding/embedding.h>
函数	void text_embedding_by_image_model_async(ImageEmbeddingSession *session, const char *text, ImageEmbeddingResultCallback callback, void *callback_user_data);

描述	通过异步的方式图像向量化模型向量化文本
参数	• session: 图像向量化会话指针 • text: 文本 • callback: 结果回调函数 • callback_user_data: 用户数据
返回值	无

4.1.3.3.8 向量化图片（异步）

4.1.3.3.8.1 通过传入图片文件路径的方式

头文件	<coreai/embedding/embedding.h>
函数	void image_embedding_from_by_file_async(ImageEmbeddingSession *session, const char *file_path, ImageEmbeddingResultCallback callback, void *callback_user_data);
描述	通过异步的方式向量化图片
参数	• session: 图像向量化会话指针 • image_file: 图片路径 • callback: 结果回调函数 • callback_user_data: 用户数据
返回值	无

4.1.3.3.8.2 通过传入base64图片数据的方式

头文件	<coreai/embedding/embedding.h>
函数	void image_embedding_by_base64_image_data_async(ImageEmbeddingSession *session, const unsigned char *image_data, unsigned int image_data_length, ImageEmbeddingResultCallback callback, void *callback_user_data);
描述	通过异步的方式向量化图片
参数	• session: 图像向量化会话指针 • image_data: base64图片数据 • image_data_length: base64图片数据的长度

	• callback: 结果回调函数 • callback_user_data: 用户数据
返回值	无

4.1.3.4 结果解析

4.1.3.4.1 获取向量化结果数据

头文件	<coreai/embedding/embedding.h>
函数	float *embedding_result_get_vector_data(EmbeddingResult *result);
描述	获取向量化结果数据
参数	• result: 向量化结果的指针
返回值	float类型指针

4.1.3.4.2 获取向量化结果数据的长度

头文件	<coreai/embedding/embedding.h>
函数	int embedding_result_get_vector_length(EmbeddingResult *result);
描述	获取向量化结果数据的长度
参数	• result: 向量化结果的指针
返回值	embedding_result_get_vector_data返回float指针数据的长度

4.1.3.4.3 获取向量化错误码

头文件	<coreai/embedding/embedding.h>
函数	int embedding_result_get_error_code(EmbeddingResult *result);
描述	获取错误码
参数	• result: 向量化结果的指针
返回值	具体的错误码

4.1.3.4.4 向量化错误信息

头文件	<coreai/embedding/embedding.h>
函数	const char *embedding_result_get_error_message(EmbeddingResult *result);
描述	获取错误信息
参数	• result: 向量化结果的指针
返回值	具体的错误信息

4.1.3.4.5 销毁向量化结果

头文件	<coreai/embedding/embedding.h>
函数	void embedding_result_destroy(EmbeddingResult **result);
描述	销毁向量化结果
参数	• result: 向量化结果指针的地址
返回值	无

4.1.3.4.6 销毁模型信息结果

头文件	<coreai/embedding/embedding.h>
函数	void embedding_model_info_destroy(char *result);
描述	销毁模型信息结果
参数	• result: 模型信息结果的指针
返回值	无

4.1.3.4.7 错误码

头文件	<coreai/embedding/error.h>
枚举	typedef enum : int { COREAI_EMBEDDING_SUCESS = 0, COREAI_EMBEDDING_INPUT_ERROR, COREAI_EMBEDDING_INIT_ERROR, COREAI_EMBEDDING_CONNECTION_ERROR,

	COREAI_EMBEDDING_RUNTIME_ERROR, COREAI_EMBEDDING_ERROR_UNKNOWN = 99, } CoreAiEmbeddingErrorCode;
描述	向量化相关的错误码
成员	• COREAI_EMBEDDING_SUCESS：成功 • COREAI_EMBEDDING_INPUT_ERROR：参数错误 • COREAI_EMBEDDING_INIT_ERROR：向量化会话初始化错误 • COREAI_EMBEDDING_CONNECTION_ERROR：向量化服务连接错误 • COREAI_EMBEDDING_RUNTIME_ERROR：向量化解析runtime结果错误 • COREAI_EMBEDDING_ERROR_UNKNOWN：向量化未知错误

4.1.3.5 示例

4.1.3.5.1 先决条件

1. 仅 x86 和 arm 架构的机器上使用向量化能力；

配置 CMakeLists.txt

代码块

```
find_package(PkgConfig REQUIRED)
pkg_check_modules(Embedding REQUIRED IMPORTED_TARGET kysdk-coreai-embedding)

target_link_libraries(
    xxx
    pthread
    ${GIO_LIBRARIES}
    PkgConfig::Embedding
)
```

4.1.3.5.2 文本向量化

4.1.3.5.2.1 同步方式

代码块

```
#include <coreai/embedding/embedding.h>
#include <iostream>

void textEmbeddingSync() {
    TextEmbeddingSession *session = text_embedding_create_session();
    int initSession = text_embedding_init_session(session);
```

```
    if (initSession != 0) {
        std::cout << "init session failed" << std::endl;
    }

    // 同步接口
    EmbeddingResult *result = nullptr;
    bool sucess = text_embedding(session, "12345", &result);

    // 结果信息
    int error_code = embedding_result_get_error_code(result);
    std::cout << "error_code: " << error_code << std::endl;
    const char *error_message = embedding_result_get_error_message(result);
    std::cout << "error_message :" << error_message << std::endl;

    float *vector_result = embedding_result_get_vector_data(result);
    int len = embedding_result_get_vector_length(result);

    std::cout << "vector_result :" << std::endl;
    for (int i = 0; i < len; i++) {
        std::cout << vector_result[i] << ",";
    }
    std::cout << std::endl;

    char *info = nullptr;
    text_embedding_get_model_info(session, &info);
    std::cout << "print model info:" << info << std::endl;

    std::cout << "释放结果资源" << std::endl;
    embedding_result_destroy(&result);
    embedding_model_info_destroy(info);
    text_embedding_destroy_session(&session);
}
```

4.1.3.5.2.2 异步方式

代码块

```
#include <coreai/embedding/embedding.h>
#include <iostream>
#include <gio/gio.h>
#include <thread>

void callback(EmbeddingResult *result, void *callback_user_data) {
    // 结果信息
    int error_code = embedding_result_get_error_code(result);
    std::cout << "error_code: " << error_code << std::endl;
    const char *error_message = embedding_result_get_error_message(result);
```

```
    std::cout << "error_message :" << error_message << std::endl;

    float *vector_result = embedding_result_get_vector_data(result);
    int len = embedding_result_get_vector_length(result);

    std::cout << "vector_result :" << std::endl;
    for (int i = 0; i < len; i++) {
        std::cout << vector_result[i] << " ";
    }
    std::cout << std::endl;

    if (callback_user_data != nullptr) {
        int *a = static_cast<int *>(callback_user_data);
        std::cout << "user_data :" << *a << std::endl;
    }
}


void textEmbeddingAsync() {
    TextEmbeddingSession *session = text_embedding_create_session();
    if (text_embedding_init_session(session) != 0) {
        std::cout << "init session failed" << std::endl;
    }
    text_embedding_async(session, "热爱学习", callback, nullptr);
    auto *loop_ = g_main_loop_new(nullptr, false);
    std::thread ctrlThread([&session, loop_] {
        while (std::getchar() != '\n') {
        }
        g_main_loop_quit(loop_);
    });
    ctrlThread.detach();

    g_main_loop_run(loop_);
    text_embedding_destroy_session(&session);
}
```

4.1.3.5.3 图像向量化

4.1.3.5.3.1 同步方式

代码块

```
#include <coreai/embedding/embedding.h>
#include <iostream>
#include <filesystem>
#include <vector>
#include <fstream>
```

```
6
7   std::vector<uint8_t> readFile(const std::string &filePath) {
8       std::ifstream file(filePath, std::ios::binary | std::ios::ate);
9       if (!file.is_open()) {
10          throw std::runtime_error("Failed to open file");
11      }
12
13      std::streamsize size = file.tellg();
14      file.seekg(0, std::ios::beg);
15
16      std::vector<uint8_t> buffer(size);
17      if (file.read(reinterpret_cast<char *>(buffer.data()), size)) {
18          return buffer;
19      } else {
20          throw std::runtime_error("Failed to read file");
21      }
22  }
23  const std::string base64_chars =
24      "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
25      "abcdefghijklmnopqrstuvwxyz"
26      "0123456789+/";
27
28  std::string base64Encode(const std::vector<uint8_t> &buffer) {
29      std::string encodedData;
30      int i = 0;
31      uint8_t char_array_3[3];
32      uint8_t char_array_4[4];
33
34      while (i < buffer.size()) {
35          char_array_3[0] = buffer[i++];
36          char_array_3[1] = (i < buffer.size()) ? buffer[i++] : 0;
37          char_array_3[2] = (i < buffer.size()) ? buffer[i++] : 0;
38
39          char_array_4[0] = (char_array_3[0] & 0xfc) >> 2;
40          char_array_4[1] =
41              ((char_array_3[0] & 0x03) << 4) + ((char_array_3[1] & 0xf0) >> 4);
42          char_array_4[2] =
43              ((char_array_3[1] & 0x0f) << 2) + ((char_array_3[2] & 0xc0) >> 6);
44          char_array_4[3] = (char_array_3[2] & 0x3f);
45
46          for (int j = 0; (j < 4); ++j) {
47              encodedData += base64_chars[char_array_4[j]];
48          }
49      }
50
51      while ((encodedData.size() % 4) != 0) {
52          encodedData += '=';
```

```
    }

    return encodedData;
}


void imageEmbeddingSync() {
    ImageEmbeddingSession *session = image_embedding_create_session();

    if (image_embedding_init_session(session) != 0) {
        std::cout << "init session failed" << std::endl;
    }
    namespace fs = std::filesystem;
    fs::path dir = fs::path(__FILE__).parent_path();
    fs::path path = dir / "微信图片_20240709181353.jpg";

    std::string imagePath = path.string();
    std::vector<uint8_t> imageData = readFile(imagePath);
    std::string base64Data = base64Encode(imageData);
    unsigned int length = base64Data.length();
    const char *charPtr = base64Data.c_str();
    const unsigned char *ucharPtr =
        reinterpret_cast<const unsigned char *>(charPtr);

#if 1  // 同步向量化文本
    EmbeddingResult *result = nullptr;
    text_embedding_by_image_model(session, "do you love working?", &result);
#elif 0
    // 同步向量化图片
    EmbeddingResult *result = nullptr;
    image_embedding_by_image_file(session, path.string().c_str(),
                                  &result);  // 换成自己路径的图片
#else
    // 同步向量化base64图片
    EmbeddingResult *result = nullptr;
    image_embedding_by_base64_image_data(session, ucharPtr, length, &result);
```

4.1.3.5.3.2 异步方式

代码块

```
#include <coreai/embedding/embedding.h>
#include <iostream>
#include <gio/gio.h>
#include <thread>
#include <vector>
#include <fstream>

```

```cpp
void callback(EmbeddingResult *result, void *callback_user_data) {
    // 结果信息
    int error_code = embedding_result_get_error_code(result);
    std::cout << "error_code: " << error_code << std::endl;
    const char *error_message = embedding_result_get_error_message(result);
    std::cout << "error_message :" << error_message << std::endl;

    float *vector_result = embedding_result_get_vector_data(result);
    int len = embedding_result_get_vector_length(result);

    std::cout << "vector_result :" << std::endl;
    for (int i = 0; i < len; i++) {
        std::cout << vector_result[i] << " ";
    }
    std::cout << std::endl;

    if (callback_user_data != nullptr) {
        int *a = static_cast<int *>(callback_user_data);
        std::cout << "user_data :" << *a << std::endl;
    }
}

std::vector<uint8_t> readFile(const std::string &filePath) {
    std::ifstream file(filePath, std::ios::binary | std::ios::ate);
    if (!file.is_open()) {
        throw std::runtime_error("Failed to open file");
    }

    std::streamsize size = file.tellg();
    file.seekg(0, std::ios::beg);

    std::vector<uint8_t> buffer(size);
    if (file.read(reinterpret_cast<char *>(buffer.data()), size)) {
        return buffer;
    } else {
        throw std::runtime_error("Failed to read file");
    }
}
const std::string base64_chars =
    "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    "abcdefghijklmnopqrstuvwxyz"
    "0123456789+/";

std::string base64Encode(const std::vector<uint8_t> &buffer) {
    std::string encodedData;
    int i = 0;
    uint8_t char_array_3[3];
```

```
    uint8_t char_array_4[4];

    while (i < buffer.size()) {
        char_array_3[0] = buffer[i++];
        char_array_3[1] = (i < buffer.size()) ? buffer[i++] : 0;
        char_array_3[2] = (i < buffer.size()) ? buffer[i++] : 0;

        char_array_4[0] = (char_array_3[0] & 0xfc) >> 2;
        char_array_4[1] =
            ((char_array_3[0] & 0x03) << 4) + ((char_array_3[1] & 0xf0) >> 4);
        char_array_4[2] =
            ((char_array_3[1] & 0x0f) << 2) + ((char_array_3[2] & 0xc0) >> 6);
        char_array_4[3] = (char_array_3[2] & 0x3f);

        for (int j = 0; (j < 4); ++j) {
            encodedData += base64_chars[char_array_4[j]];
        }
    }

    while ((encodedData.size() % 4) != 0) {
        encodedData += '=';
    }

    return encodedData;
}

void imageEmbeddingAsync() {
    ImageEmbeddingSession *session = image_embedding_create_session();

    if (image_embedding_init_session(session) != 0) {
        std::cout << "init session failed" << std::endl;
    }

    namespace fs = std::filesystem;
```

4.2 生成式 AI 能力接口

4.2.1 文本生成

1. 支持文本生成、文本对话；
2. 内置多种默认提示词；
3. 支持云端、端侧和自定义的模型，具体可在“设置->AI 模块管理”中进行配置。

4.2.1.1 开发环境部署

```
sudo apt install libkysdk-genai-nlp-dev
```

4.2.1.2 会话

4.2.1.2.1 创建会话

头文件	<genai/text/chat.h>
函数	GenAiTextSession *genai_text_create_session()
描述	创建文本生成会话
参数	无
返回值	文本生成会话的指针

4.2.1.2.2 初始化会话

头文件	<genai/text/chat.h>
函数	int genai_text_init_session(GenAiTextSession *session)
描述	初始化会话
参数	session: 文本生成会话指针
返回值	成功时返回 0，否则返回具体的错误码

4.2.1.2.3 销毁会话

头文件	<genai/text/chat.h>
函数	void genai_text_destroy_session(GenAiTextSession **session)
描述	销毁文本生成会话
参数	session: 文本生成会话指针的地址
返回值	无

4.2.1.2.4 设置结果回调函数

头文件	<genai/text/chat.h>
函数	void genai_text_result_set_callback(GenAiTextSession *session, ChatResultCallback callback, void *user_data)

描述	设置对话结果回调函数
参数	• session: 文本生成会话的指针 • callback: 结果回调函数 • user_data: 用户的数据
返回值	无

4.2.1.2.5 设置模型配置

头文件	<genai/text/chat.h>
函数	void genai_text_set_model_config(GenAiTextSession *session, ChatModelConfig *config)
描述	设置模型配置
参数	• session: 文本生成会话指针 • config: 模型配置
返回值	无

4.2.1.2.6 文本生成

头文件	<genai/text/chat.h>
函数	void genai_text_generate_content_async(GenAiTextSession *session, const char *prompt)
描述	根据提示内容生成文本
参数	• session: 文本生成会话指针 • prompt: 输入提示词文本
返回值	无

4.2.1.2.7 文本对话 - 支持缓存历史消息

头文件	<genai/text/chat.h>
函数	void genai_text_chat_async(GenAiTextSession *session, const char *question)

描述	进行文本对话，支持缓存历史消息
参数	- session: 文本生成会话 - question: 具体的问题
返回值	无

4.2.1.2.8 文本对话 - 不支持缓存历史消息

头文件	<genai/text/chat.h>
函数	void genai_text_chat_with_history_messages_async(GenAiTextSession *session, ChatMessage *history_messages)
描述	进行文本对话，需要传入历史消息
参数	• session: 文本生成会话 • history_messages: 历史消息
返回值	无

4.2.1.2.9 设置提示词 - 自定义提示词

头文件	<genai/text/chat.h>
函数	void genai_text_set_chat_system_prompt(GenAiTextSession *session, const char *prompt)
描述	设置系统提示词
参数	• session: 文本生成会话 • prompt: 具体的系统提示词
返回值	无

4.2.1.2.10 设置提示词 - 使用系统内置提示词

头文件	<genai/text/chat.h>
函数	void genai_text_set_chat_system_prompt_id(GenAiTextSession *session, PromptId prompt_id)
描述	设置系统提示词 id，使用内置提示词

参数	• session: 文本生成会话 • prompt_id: 提示词 id
返回值	无

4.2.1.2.11 清除历史消息

头文件	<genai/text/chat.h>
函数	void genai_text_clear_chat_history_messages(GenaiTextSession *session)
描述	清除对话历史消息
参数	• session: 文本生成会话
返回值	无

4.2.1.2.12 停止会话

头文件	<genai/text/chat.h>
函数	void genai_text_stop_chat(GenaiTextSession *session)
描述	停止对话
参数	• session: 文本生成会话
返回值	无

4.2.1.3 模型参数配置

4.2.1.3.1 创建模型配置实例

头文件	<genai/text/config.h>
函数	ChatModelConfig *chat_model_config_create()
描述	创建模型配置实例
参数	无
返回值	模型配置实例指针

4.2.1.3.2 销毁模型配置实例

头文件	<genai/text/config.h>
函数	void chat_model_config_destroy(ChatModelConfig **config)
描述	销毁模型配置实例
参数	• config: 模型配置的二级指针
返回值	无

4.2.1.3.3 设置使用的模型

头文件	<genai/text/config.h>
函数	void chat_model_config_set_name(ChatModelConfig *config, const char *model_name)
描述	设置模型的名称
参数	• config: 模型配置实例指针 • model_name: 模型的名称
返回值	无

4.2.1.3.4 设置模型的参数

头文件	<genai/text/config.h>
函数	void chat_model_config_set_top_k(ChatModelConfig *config, double top_k)
描述	设置模型的 top_k 参数
参数	• config: 模型配置实例的指针 • top_k: top_k 参数的数值
返回值	无

4.2.1.3.5 设置模型的部署类型

1. 指定要使用的模型的部署类型；
2. 如果未指定部署类型，则会根据“设置->AI 模块管理”中配置的优先级进行选择。

头文件	<genai/text/config.h>

函数	void chat_model_config_set_deploy_type(ChatModelConfig *config, ModelDeployType type)
描述	设置模型的部署类型
参数	• config: 模型配置实例的指针 • type: 模型类型
返回值	无

4.2.1.3.6 设置模型的名称

1. 名称是是指“设置->AI 模块管理”中配置的名称；
2. 如果同时指定了部署类型和名称，会优先匹配名称，如果无法匹配，则匹配对应的部署类型；
3. 如果未指定名称，则会根据“设置->AI 模块管理”中配置的优先级进行选择。

头文件	<genai/text/config.h>
函数	void chat_model_config_set_name(ChatModelConfig *config, const char *name);
描述	设置模型的名称
参数	• config: 模型配置实例的指针 • name: 模型的名称
返回值	无

4.2.1.4 结果解析

4.2.1.4.1 获取模型生成的消息

头文件	<genai/text/result.h>
函数	const char *chat_result_get_assistant_message(ChatResult *result)
描述	获取模型生成的文本结果
参数	• result: 文本生成结果实例指针
返回值	文本字符串

4.2.1.4.2 获取生成结束的原因

头文件	<genai/text/result.h>

函数	const char *chat_result_get_finish_reason_message(ChatResult *result)
描述	获取对话结束的原因
参数	• result: 文本生成的结果的指针
返回值	具体的原因

4.2.1.4.3 获取生成结果的错误码

头文件	<genai/text/result.h>
函数	int chat_result_get_error_code(ChatResult *result)
描述	获取具体的错误码
参数	• result: 文本生成的结果的指针
返回值	具体的错误码

4.2.1.4.4 获取生成结果的具体错误信息

头文件	<genai/text/result.h>
函数	const char *chat_result_get_error_message(ChatResult *result)
描述	获取具体的错信息
参数	• result: 文本生成的结果的指针
返回值	具体错误信息

4.2.1.4.5 获取生成结果的结束标志

头文件	<genai/text/result.h>
函数	const char *chat_result_get_is_end(ChatResult *result)
描述	获取是否是最后结果的标志
参数	• result: 文本生成的结果的指针
返回值	是否为结束消息

4.2.1.5 错误码

通用错误码可参考 4.3 节，文本生成专用错误码如下：

头文件	<genai/text/error.h>
枚举	typedef enum { NLP_INPUT_INVALID = 100, NLP_PARAM_ERROR } GenAiTextErrorCode;
描述	文本生成相关的错误码
成员	• NLP_INPUT_INVALID：输入文本无效 • NLP_PARAM_ERROR：参数错误

4.2.1.6 示例

4.2.1.6.1 前提条件

1. 仅 x86 和 arm 架构的机器上可使用端侧模型；
2. 如果想使用云端模型或者自定义模型需要在“设置->AI 模块管理”中进行配置。

配置 CMakeLists.txt

代码块

```
find_package(PkgConfig REQUIRED)
pkg_check_modules(KYAINLP kysdk-genai-nlp)
include_directories(${KYAINLP_INCLUDE_DIRS})

target_link_libraries(
    xxx
    pthread
    ${GIO_LIBRARIES}
    ${KYAINLP_LIBRARIES}
)
```

4.2.1.6.2 回调函数设置

后续测试如果没有特殊说明，都使用该回调

```
void callback(ChatResult *result, void *user_data) {
    auto getBool = [](bool value) { return value ? "true" : "false"; };
```

```
    fprintf(stdout, "%s\n", chat_result_get_assistant_message(result));
    fprintf(stdout, "%s\n", chat_result_get_finish_reason_message(result));
    fprintf(stdout, "%d\n", chat_result_get_error_code(result));
    fprintf(stdout, "%s\n", chat_result_get_error_message(result));
    fprintf(stdout, "%s\n", getBool(chat_result_get_is_end(result)));
    if (user_data != nullptr) {
        int *a = static_cast<int *>(user_data);
        fprintf(stdout, "%d\n", *a);
    }
}
```

4.2.1.6.3 基本会话

```
// 此样例未设置提示词

#include <gio/gio.h>
#include <gio/giotypes.h>

#include <filesystem>
#include <fstream>
#include <iostream>
#include <thread>
#include <vector>

#include <genai/text/chat.h>

void chat() {
    GMainLoop *pMainLoop = g_main_loop_new(nullptr, false);

    std::thread ctrlThread([pMainLoop] {
        g_main_loop_run(pMainLoop);
        g_main_loop_unref(pMainLoop);
    });
    ctrlThread.detach();

    ChatModelConfig *config = chat_model_config_create();
    chat_model_config_set_name(config, "百度-ERNIE-Bot-4");
    chat_model_config_set_top_k(config, 0.5);
    chat_model_config_set_deploy_type(config, ModelDeployType::PublicCloud);

    GenAiTextSession *session = genai_text_create_session();
    genai_text_set_model_config(session, config);

    genai_text_init_session(session);
    int a = 100;
```

```
    genai_text_result_set_callback(session, callback, &a);
    genai_text_chat_async(session, "一加一等于几");

    while (std::getchar() != '\n') {
    }

    genai_text_chat_async(session, "你说的不对");

    while (std::getchar() != '\n') {
    }
    genai_text_stop_chat(session);
    genai_text_destroy_session(&session);
    chat_model_config_destroy(&config);
    g_main_loop_quit(pMainLoop);
}
```

4.2.1.6.4 使用历史消息会话

```
#include <gio/gio.h>
#include <gio/giotypes.h>

#include <filesystem>
#include <fstream>
#include <iostream>
#include <thread>
#include <vector>

#include <genai/text/chat.h>

void chatHistoryMessages() {
    GMainLoop *pMainLoop = g_main_loop_new(nullptr, false);

    std::thread ctrlThread([pMainLoop] {
        g_main_loop_run(pMainLoop);
        g_main_loop_unref(pMainLoop);
    });
    ctrlThread.detach();

    ChatModelConfig *config = chat_model_config_create();
    chat_model_config_set_name(config, "百度-ERNIE-Bot-4");
    chat_model_config_set_top_k(config, 0.5);
    chat_model_config_set_deploy_type(config, ModelDeployType::PublicCloud);

    GenAiTextSession *session = genai_text_create_session();
    genai_text_set_model_config(session, config);
```

```
28
29      genai_text_init_session(session);
30      genai_text_result_set_callback(session, callback, nullptr);
31
32      ChatMessage *chatMessage = chat_message_create();
33      chat_message_add_system_message(chatMessage, "");
34      chat_message_add_user_message(chatMessage, "一加一等于几");
35      chat_message_add_system_message(chatMessage,
36                                      "这是一个非常基础的数学问题，"
37                                      "、涉及到的是加法运算。题目问"
38                                      "的是1+1等于几。\n\n在数学中，"
39                                      "加法是一种基本的运算方式，表"
40                                      "示两个数量的和。当我们把两个1"
41                                      "加在一起时，就是在计算这两个"
42                                      "数量的总和。\n\n所以，1 + 1 ="
43                                      " 2。\n\n因此，答案是2。这个问"
44                                      "题非常直接，没有涉及到复杂的"
45                                      "数学概念或技巧，只需要理解加"
46                                      "法的基本定义即可。");
47      chat_message_add_user_message(chatMessage, "你说的不对");
48
49      genai_text_chat_with_history_messages_async(session, chatMessage);
50
51      while (std::getchar() != '\n') {
52      }
53      chat_message_destroy(&chatMessage);
54      genai_text_stop_chat(session);
55      genai_text_destroy_session(&session);
56      chat_model_config_destroy(&config);
57      g_main_loop_quit(pMainLoop);
58  }
```

4.2.1.6.5 清理消息

```
1   #include <gio/gio.h>
2   #include <gio/giotypes.h>
3
4   #include <filesystem>
5   #include <fstream>
6   #include <iostream>
7   #include <thread>
8   #include <vector>
9
10  #include <genai/text/chat.h>
11
```

```
void clearChatMessage() {
    GMainLoop *pMainLoop = g_main_loop_new(nullptr, false);

    std::thread ctrlThread([pMainLoop] {
        g_main_loop_run(pMainLoop);
        g_main_loop_unref(pMainLoop);
    });
    ctrlThread.detach();

    ChatModelConfig *config = chat_model_config_create();
    chat_model_config_set_name(config, "百度-ERNIE-Bot-4");
    chat_model_config_set_top_k(config, 0.5);
    chat_model_config_set_deploy_type(config, ModelDeployType::PublicCloud);

    GenAiTextSession *session = genai_text_create_session();
    genai_text_set_model_config(session, config);

    genai_text_init_session(session);
    genai_text_result_set_callback(session, callback, nullptr);
    genai_text_chat_async(session, "一加一等于几");

    while (std::getchar() != '\n') {
    }

    genai_text_clear_chat_history_messages(session);

    while (std::getchar() != '\n') {
    }

    genai_text_chat_async(session, "你说的不对");

    while (std::getchar() != '\n') {
    }

    genai_text_stop_chat(session);
    genai_text_destroy_session(&session);
    chat_model_config_destroy(&config);
    g_main_loop_quit(pMainLoop);
}
```

4.2.1.6.6 内容生成

```
#include <gio/gio.h>
#include <gio/giotypes.h>

```

```cpp
#include <filesystem>
#include <fstream>
#include <iostream>
#include <thread>
#include <vector>

#include <genai/text/chat.h>

void generateContent() {
    GMainLoop *pMainLoop = g_main_loop_new(nullptr, false);

    std::thread ctrlThread([pMainLoop] {
        g_main_loop_run(pMainLoop);
        g_main_loop_unref(pMainLoop);
    });
    ctrlThread.detach();

    ChatModelConfig *config = chat_model_config_create();
    chat_model_config_set_top_k(config, 0.5);
    // 使用云端模型
    chat_model_config_set_deploy_type(config, ModelDeployType::PublicCloud);

    // 使用端侧模型
    // chat_model_config_set_deploy_type(config, ModelDeployType::OnDevice);

    GenAiTextSession *session = genai_text_create_session();
    genai_text_set_model_config(session, config);

    genai_text_init_session(session);
    genai_text_result_set_callback(session, callback, nullptr);
    genai_text_generate_content_async(session, "今天天气不错");

    while (std::getchar() != '\n') {
    }
    genai_text_stop_chat(session);
    genai_text_destroy_session(&session);
    g_main_loop_quit(pMainLoop);
}
```

4.2.1.6.7 使用系统内置提示词对话

```cpp
#include <gio/gio.h>
#include <gio/giotypes.h>

#include <filesystem>
```

```cpp
#include <fstream>
#include <iostream>
#include <thread>
#include <vector>

#include <genai/text/chat.h>

void chatSystemPromptId(PromptId promptId, const std::string &question) {
    GMainLoop *pMainLoop = g_main_loop_new(nullptr, false);

    std::thread ctrlThread([pMainLoop] {
        g_main_loop_run(pMainLoop);
        g_main_loop_unref(pMainLoop);
    });
    ctrlThread.detach();

    ChatModelConfig *config = chat_model_config_create();
    chat_model_config_set_name(config, "百度-ERNIE-Bot-4");
    chat_model_config_set_top_k(config, 0.5);
    chat_model_config_set_deploy_type(config, ModelDeployType::PublicCloud);

    GenAiTextSession *session = genai_text_create_session();
    genai_text_set_model_config(session, config);

    genai_text_init_session(session);
    genai_text_result_set_callback(session, callback, nullptr);
    genai_text_set_chat_system_prompt_id(session, promptId);
    genai_text_chat_async(session, question.c_str());

    while (std::getchar() != '\n') {
    }
    genai_text_stop_chat(session);
    genai_text_destroy_session(&session);
    g_main_loop_quit(pMainLoop);
}

void chatSystemPromptId_SUMMARY() {
    chatSystemPromptId(
        SUMMARY,
        "大模型是人工智能领域的热门研究方向。专家认为，人工智能进"
        "入产业级大模型时代。大模型将是未来一段时间科技领域里面最重要的事情之一"
        "。大模型将开启人工智能的“大一统时代”。");
}

void chatSystemPromptId_TEXT_EXPANSION() {
    chatSystemPromptId(TEXT_EXPANSION, "今天天气不错");
}
```

```
52
53  void chatSystemPromptId_TRANSLATE_CHINESE_TO_ENGLISH() {
54      chatSystemPromptId(TRANSLATE_CHINESE_TO_ENGLISH,"今天天气不错");
55  }
```

4.2.2 图像生成

1. 根据文本描述生图片；
2. 支持多种风格；
3. 支持多种分辨率；
4. 支持多种语言；
5. 具体参数依赖云端服务；
6. 目前仅支持云端服务。

4.2.2.1 开发环境部署

`sudo apt install libkysdk-genai-vision-dev`

4.2.2.2 会话

4.2.2.2.1 创建会话

头文件	<genai/vision/image.h>
函数	GenAiImageSession *genai_image_create_session()
描述	创建图片生成会话
参数	无
返回值	图片生成会话的指针

4.2.2.2.2 初始化会话

头文件	<genai/vision/image.h>
函数	int genai_image_init_session(GenAiImageSession *session)
描述	初始化会话
参数	session: 图片生成会话指针
返回值	成功时返回 0，否则返回具体的错误码

4.2.2.2.3 销毁会话

头文件	<genai/vision/image.h>
函数	void genai_image_destroy_session(GenAiImageSession **session)
描述	销毁图片生成会话
参数	session: 图片生成会话指针的地址
返回值	无

4.2.2.2.4 设置图像生成的相关配置

头文件	<genai/vision/image.h>
函数	void genai_image_set_config(GenAiImageSession *session, ImageConfig *config)
描述	设置图像生成的相关配置
参数	• session: 文本生成会话指针 • config: 配置实例的指针
返回值	无

4.2.2.2.5 设置结果回调函数

头文件	<genai/vision/image.h>
函数	void genai_image_result_set_callback(GenAiTextSession *session, ImageResultCallback callback, void *user_data)
描述	设置对话结果回调函数
参数	• session: 图片生成会话的指针 • callback: 结果回调函数 • user_data: 用户的数据
返回值	无

4.2.2.2.6 获取支持图片风格

头文件	<genai/vision/image.h>

函数	const char **genai_image_get_supported_image_style(GenAiImageSession *session, int *number)
描述	获取支持的图片样风格，如古风，二次元等
参数	• session: 图片生成会话的指针 • number: 支持的图片风格数目，输出参数
返回值	返回字符串数组的首地址（const char**）

4.2.2.2.7 获取支持的图片尺寸

头文件	<genai/vision/image.h>
函数	const ImageSize *genai_image_get_supported_image_size(GenAiImageSession *session, int *number)
描述	获取支持的图片尺寸，如 1280*720，1920*1080 等
参数	• session: 图片生成会话的指针 • number: 支持的图片尺寸的数量
返回值	返回 ImageSize 数组的地址首地址，ImageSize 包含两个参数（width 和 height）

4.2.2.2.8 获取支持生成图片的数量

头文件	<genai/vision/image.h>
函数	int genai_image_get_supported_image_number(GenAiImageSession *session)
描述	获取支持生成图片数量
参数	• session: 图片生成会话的指针
返回值	返回支持生成图片数量

4.2.2.2.9 生成图片

头文件	<genai/vision/image.h>
函数	void genai_image_generate_image_async(GenAiImageSession *session, const char *prompt)
描述	根据提示词生成图片

参数	• session: 图片生成会话的指针
返回值	无

4.2.2.3 图片配置

4.2.2.3.1 图片尺寸结构体

头文件	<genai/vision/imageconfig.h>
结构体名称	typedef struct _ImageSize { int width; int height; } ImageSize
描述	图片尺寸
公有成员变量：width	类型：int；描述：宽度
公有成员变量：height	类型：int；描述：高度

4.2.2.3.2 图片配置结构体创建

头文件	<genai/vision/imageconfig.h>
函数	ImageConfig *image_config_create()
描述	创建图片配置相关的结构体实例
参数	无
返回值	图片配置结构体指针

4.2.2.3.3 图片配置结构体销毁

头文件	<genai/vision/imageconfig.h>
函数	void image_config_destroy(ImageConfig **config)
描述	销毁图片配置结构体
参数	• config: 图片配置结构体指针的地址

返回值	无

4.2.2.3.4 图片配置结构体设置生成数量

头文件	<genai/vision/imageconfig.h>
函数	void image_config_set_generation_number(ImageConfig *config, int number)
描述	图片配置结构体设置生成数量
参数	• config: 图片配置结构体指针 • number: 生成数量
返回值	无

4.2.2.3.5 图片配置结构体设置风格

头文件	<genai/vision/imageconfig.h>
函数	void image_config_set_style(ImageConfig *config, const char *style)
描述	图片配置结构体设置风格
参数	• config: 图片配置结构体指针 • style: 风格（如古风、二次元等）
返回值	无

4.2.2.3.6 图片配置结构体设置图片尺寸

头文件	<genai/vision/imageconfig.h>
函数	void image_config_set_size(ImageConfig *config, ImageSize image_size)
描述	图片配置结构体设置图片尺寸
参数	• config: 图片配置结构体指针 • image_size: 图片尺寸
返回值	无

4.2.2.4 错误码

通用错误码参考 4.3 节，文生图专有错误码如下：

头文件	<genai/vision/error.h>
枚举	typedef enum { VISION_INPUT_TEXT_LENGTH_INVALID = 100， VISION_IMAGE_STYLE_INVALID， VISION_IMAGE_SIZE_INVALID， VISION_IMAGE_NUMBER_INVALID， VISION_IMAGE_GENERATION_BLOCKED， VISION_IMAGE_GENERATION_FAILED， } GenAiVisionErrorCode;
描述	文本生成相关的错误码
成员	• VISION_INPUT_TEXT_LENGTH_INVALID：输入的提示词文本过长 • VISION_IMAGE_STYLE_INVALID：不支持的风格 • VISION_IMAGE_SIZE_INVALID：不支持的图片大小 • VISION_IMAGE_NUMBER_INVALID：不支持的图片数量 • VISION_IMAGE_GENERATION_BLOCKED：生成的图片未过审 • VISION_IMAGE_GENERATION_FAILED：生成图片失败

4.2.2.5 结果解析

4.2.2.5.1 获取生成图片格式

头文件	<genai/vision/imageresult.h>
函数	ImageFormat image_result_get_format(VisionImageResult *image_result)
描述	获取图片格式（jpg、png 等）
参数	• image_result: 图片配结果结构体指针
返回值	图片格式（jpg、png 等）

4.2.2.5.2 获取生成图片尺寸

头文件	<genai/vision/imageresult.h>

函数	ImageSize image_result_get_size(VisionImageResult *image_result)
描述	获取图片尺寸
参数	• image_result: 图片结果结构体指针
返回值	图片尺寸结构体

4.2.2.5.3 获取生成图片总数量

头文件	<genai/vision/imageresult.h>
函数	int image_result_get_total(VisionImageResult *image_result)
描述	获取生成图片总数量
参数	• image_result: 图片结果结构体指针
返回值	生成图片总数量

4.2.2.5.4 获取生成图片序号

头文件	<genai/vision/imageresult.h>
函数	int image_result_get_index(VisionImageResult *image_result)
描述	获取生成图片序号
参数	• image_result: 图片结果结构体指针
返回值	生成图片序号

4.2.2.5.5 获取生成图片数据

头文件	<genai/vision/imageresult.h>
函数	const uint8_t *image_result_get_data(VisionImageResult *image_result, int *data_length)
描述	获取生成图片数据
参数	• image_result: 图片结果结构体指针 • data_length: 图片数据长度
返回值	图片数据首地址

4.2.2.5.6 获取生成图片结果错误码

头文件	<genai/vision/imageresult.h>
函数	int image_result_get_error_code(VisionImageResult *image_result)
描述	获取图片尺寸获取生成图片结果错误码
参数	• image_result: 图片结果结构体指针
返回值	生成图片结果错误码

4.2.2.5.7 获取生成图片结果错误信息

头文件	<genai/vision/imageresult.h>
函数	const char *image_result_get_error_message(VisionImageResult *image_result)
描述	获取生成图片结果错误信息
参数	• image_result: 图片结果结构体指针
返回值	生成图片结果错误信息

4.2.2.6 代码示例

4.2.2.6.1 前提条件

需要在“设置->AI 模块管理”中配置文生图相关的账号。

4.2.2.6.2 示例

配置 CMakeLists.txt

代码块

```
find_package(PkgConfig REQUIRED)
pkg_check_modules(KYAIVISION kysdk-genai-vision)
include_directories(${KYAIVISION_INCLUDE_DIRS})

target_link_libraries(
    xxx
    pthread
    ${GIO_LIBRARIES}
    ${KYAIVISION_LIBRARIES}
)
```

具体 demo：

```cpp
#include <cstdio>
#include <filesystem>
#include <fstream>
#include <iostream>
#include <thread>
#include <vector>
#include <genai/vision/image.h>
#include <gio/gio.h>

static void writeBinaryDataToFile(const std::string &filename,
                                  const std::vector<char> &data) {
    std::ofstream outputFile(filename, std::ios::out | std::ios::binary | std::ios::trunc);
    if (!outputFile.is_open()) {
        std::cerr << "Failed to open file for writing." << std::endl;
        return;
    }

    // 写入数据到文件
    outputFile.write(data.data(), data.size());

    // 检查是否写入成功
    if (!outputFile.good()) {
        std::cerr << "Error occurred while writing to file." << std::endl;
    } else {
        std::cout << "Binary data has been written to file: " << filename
                  << std::endl;
    }

    // 关闭文件
    outputFile.close();
}

void callback(VisionImageResult *image_data, void *user_data) {
    int imageDataLength;
    const uint8_t* imageData = image_result_get_data(image_data, &imageDataLength);
    int imageWidth = image_result_get_size(image_data).width;
    int imageHeight = image_result_get_size(image_data).height;
    ImageFormat format = image_result_get_format(image_data);
    int index = image_result_get_index(image_data);
    int total = image_result_get_total(image_data);

    int errorcode = image_result_get_error_code(image_data);
```

```
43      std::string errorMsg = image_result_get_error_message(image_data);
44
45
46      std::vector<char> imagedata { imageData, imageData + imageDataLength };
47      writeBinaryDataToFile("../test3.png", imagedata);
48
49      int userData = *(int*)user_data;
50      std::cout << "length " << imageDataLength
51                << " width " << imageWidth
52                << " height " << imageHeight
53                << " format " << format << std::endl;
54      std::cout << "index " << index
55                << " total " << total
56                << " errorCode " << errorcode
57                << " errorMag " << errorMsg
58                << " userdata " << userData << std::endl;
59  }
60
61  void test() {
62      auto *loop_ = g_main_loop_new(nullptr, false);
63
64      GenAiImageSession *session = genai_image_create_session();
65      int initRet = genai_image_init_session(session);
66      fprintf(stderr, "init return value %i\n", initRet);
67
68      ImageConfig *config = image_config_create();
69      image_config_set_generation_number(config, 1);
70      image_config_set_style(config, "写实风格");
71      image_config_set_size(config, ImageSize{1280, 720});
72      genai_image_set_config(session, config);
73
74      bool imageNumber = genai_image_get_supported_image_number(session);
75      std::cout << "supported image number " << imageNumber << std::endl;
76
77      int sizeNumber;
78      const ImageSize* imageSize =
    genai_image_get_supported_image_size(session, &sizeNumber);
79      int i = 0;
80      while (i < sizeNumber) {
81          std::cout << "supported image size width " << imageSize[i].width
82                    << " height " << imageSize[i].height << std::endl;
83          ++i;
84      }
85
86      int styleNumber;
87      int j = 0;
```

```
    const char** imageStyle = genai_image_get_supported_image_style(session, &styleNumber);
    while (j < styleNumber) {
        std::cout << "supported image style " << imageStyle[j] << std::endl;
        ++j;
    }

    int *a = new int();
    *a = 100;
    genai_image_result_set_callback(session, callback, a);

    genai_image_generate_image_async(session, "生成一张小狗的图片");

    g_main_loop_run(loop_);
}
```

4.3 通用错误码

```
typedef enum {
    // 未发生错误
    AISDK_NO_ERROR = 0,
    // 网络错误，比如网络断开或者网速较慢等
    AISDK_NET_ERROR,
    // 鉴权错误，比如云端服务的账号信息填写有误
    AISDK_AUTHENTICATION_FAILURE,
    // 运行时服务错误，后端服务发生错误，可重试或者进行反馈
    AISDK_RUNTIME_ERROR,
    // 请求次数过多，多见于云端服务的场景
    AISDK_TOO_MANY_REQUESTS,
    // 云端服务错误
    AISDK_SERVICE_ERROR,
    // 云端服务超时
    AISDK_SERVICE_TIMEOUT,
    // 参数错误，请求格式错误
    AISDK_BAD_REQUEST,
    // 模型运行失败
    AISDK_MODEL_RUN_FAILED,
    // 模型运行超时
    AISDK_MODEL_RUN_TIME_OUT,
    // 未找到可用的模型
    AISDK_MODEL_NOT_FOUND,
} AiSdkCommonErrorCode;
```