**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：

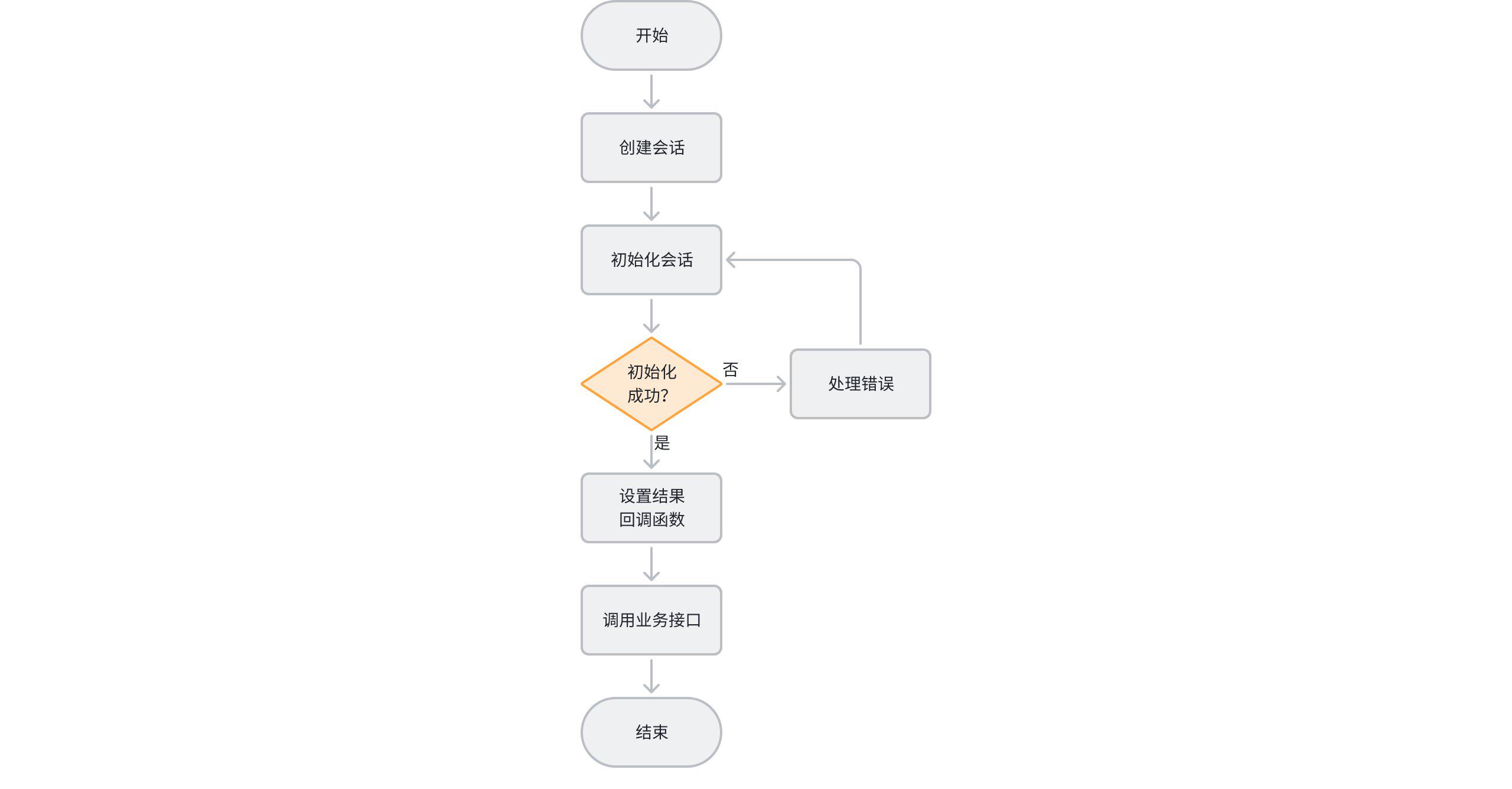
|  |  |
| --- | --- |
| 整机型号 | 具体配置 |
| 联想开天M90f G1s | CPU: D3000内存：16G独立显卡：Arise-GT10C0t硬盘：512G |
| 联想开天M90h G1t | CPU: Hygon C86-3G (OPN:3350) 内存：16GB独立显卡：Arise-GT10C0t硬盘：512GB |
| 联想开天 P90z G1t | CPU: ZHAOXIN KaiXian KX-7000 内存：16GB 独立显卡：Arise-GT10C0t 硬盘：1TB |

1. 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 |

1. 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。 |

1. 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 |

1. 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 |

1. 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 |

1. 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

|  |
| --- |
| C++ 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

|  |
| --- |
| C++ #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

|  |
| --- |
| C++ #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 {};  }   file.seekg(0, std::ios::end);  std::streampos fileSize = file.tellg();  file.seekg(0, std::ios::beg);  std::vector<char> imageData(fileSize);  file.read(reinterpret\_cast*<*char *\*>(imageData.data())*, fileSize);  return imageData; }  void test02\_OcrFromData() {  namespace fs = std::filesystem;  if (not fs::exists(TEST\_DATA\_PATH)) {  fprintf(stderr, "error\n");  return;  }   const char \*userData = "Test genai vision from image data\n";  const std::vector<char> imageData = readImageData(TEST\_DATA\_PATH);   TextRecognitionSession \*session = text\_recognition\_create\_session();   TextRecognitionModelConfig \*config = text\_recognition\_model\_config\_create();   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\_data\_async(  *session*, imageData.data(), imageData.size());   GMainLoop \*pMainLoop = g\_main\_loop\_new(nullptr, false);   std::thread ctrlThread([&session, &config, pMainLoop] {  while (std::getchar() != '\n') {  }  text\_recognition\_model\_config\_destroy(*&config*);  text\_recognition\_destroy\_session(*&session*);  g\_main\_loop\_quit(*pMainLoop*);  });  ctrlThread.detach();  g\_main\_loop\_run(*pMainLoop*);  g\_main\_loop\_unref(*pMainLoop*); } |

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 |

1. 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 |

1. 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 |

1. 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 |

1. 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 |

1. 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 |

1. 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 |

1. 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 |

1. 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 |

1. 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 |

1. 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 |

1. 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 |

1. 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 |

1. 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 |

1. 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 |

1. 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

|  |
| --- |
| C++ 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：

|  |
| --- |
| c++ #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**

|  |
| --- |
| c++ #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();  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\_synthesizer\_set\_model\_config(*synSession*, *modelconfig*);   speech\_synthesizer\_result\_set\_callback(*synSession*, onSynthesisResult, nullptr);  speech\_synthesizer\_init\_session(*synSession*);   audio\_config\_set\_input\_audio\_rate(*synthesizerConfig*, 16000);  speech\_synthesizer\_set\_audio\_config(*synSession*, *synthesizerConfig*);   speech\_synthesizer\_synthesize\_text\_async(*synSession*, "你好", 100);   int stopErrorCode = speech\_synthesizer\_stop\_speaking(*synSession*);   g\_main\_loop\_run(*pMainLoop*);  g\_main\_loop\_unref(*pMainLoop*); } |

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

|  |
| --- |
| C++ 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**

|  |
| --- |
| c++ #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);   const char \*error\_message = embedding\_result\_get\_error\_message(result);    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 << "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**

|  |
| --- |
| c++ #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);   const char \*error\_message = embedding\_result\_get\_error\_message(result);    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);   } }   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**

|  |
| --- |
| Plain Text #include <coreai/embedding/embedding.h> #include <iostream> #include <filesystem> #include <vector> #include <fstream>  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 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); #endif  // 结果信息  int error\_code = embedding\_result\_get\_error\_code(result);   const char \*error\_message = embedding\_result\_get\_error\_message(result);    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;  image\_embedding\_get\_model\_info(session, &info);    embedding\_result\_destroy(&result);  embedding\_model\_info\_destroy(info);  image\_embedding\_destroy\_session(&session); } |

4.1.3.5.3.2 **异步方式**

|  |
| --- |
| Plain Text #include <coreai/embedding/embedding.h> #include <iostream> #include <gio/gio.h> #include <thread> #include <vector> #include <fstream>  void callback(EmbeddingResult \*result, void \*callback\_user\_data) {  // 结果信息  int error\_code = embedding\_result\_get\_error\_code(result);   const char \*error\_message = embedding\_result\_get\_error\_message(result);    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::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;  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  text\_embedding\_by\_image\_model\_async(session,"do you love working?",callback,nullptr); #elif 0  image\_embedding\_from\_by\_file\_async(session, imagePath.c\_str(), callback,  nullptr); #else  image\_embedding\_by\_base64\_image\_data\_async(session, ucharPtr, length,  callback, nullptr); #endif   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\_);  image\_embedding\_destroy\_session(&session); } |

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

|  |
| --- |
| C++ 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

|  |
| --- |
| C++ 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**

|  |
| --- |
| C++ // 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**

|  |
| --- |
| c++ #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*);   genai\_text\_init\_session(*session*);  genai\_text\_result\_set\_callback(*session*, callback, nullptr);   ChatMessage \*chatMessage = chat\_message\_create();  chat\_message\_add\_system\_message(*chatMessage*, "");  chat\_message\_add\_user\_message(*chatMessage*, "一加一等于几");  chat\_message\_add\_system\_message(*chatMessage*,  "这是一个非常基础的数学问题，"  "、涉及到的是加法运算。题目问"  "的是1+1等于几。\n\n在数学中，"  "加法是一种基本的运算方式，表"  "示两个数量的和。当我们把两个1"  "加在一起时，就是在计算这两个"  "数量的总和。\n\n所以，1 + 1 ="  " 2。\n\n因此，答案是2。这个问"  "题非常直接，没有涉及到复杂的"  "数学概念或技巧，只需要理解加"  "法的基本定义即可。");  chat\_message\_add\_user\_message(*chatMessage*, "你说的不对");   genai\_text\_chat\_with\_history\_messages\_async(*session*, *chatMessage*);   while (std::getchar() != '\n') {  }  chat\_message\_destroy(*&chatMessage*);  genai\_text\_stop\_chat(*session*);  genai\_text\_destroy\_session(*&session*);  chat\_model\_config\_destroy(*&config*);  g\_main\_loop\_quit(*pMainLoop*); } |

4.2.1.6.5 **清理消息**

|  |
| --- |
| c++ #include <gio/gio.h> #include <gio/giotypes.h>  #include <filesystem> #include <fstream> #include <iostream> #include <thread> #include <vector>  #include <genai/text/chat.h>  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**

|  |
| --- |
| c++ #include <gio/gio.h> #include <gio/giotypes.h>  #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**

|  |
| --- |
| typescript #include <gio/gio.h> #include <gio/giotypes.h>  #include <filesystem> #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"); }  void chatSystemPromptId\_TRANSLATE\_CHINESE\_TO\_ENGLISH() {  chatSystemPromptId(TRANSLATE\_CHINESE\_TO\_ENGLISH,"今天天气不错"); } |

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

|  |
| --- |
| C++ 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：

|  |
| --- |
| c++ #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*);  std::string errorMsg = image\_result\_get\_error\_message(*image\_data*);    std::vector<char> imagedata { imageData, imageData + imageDataLength };  writeBinaryDataToFile("../test3.png", imagedata);   int userData = \*(int\*)user\_data;  std::cout << "length " << imageDataLength  << " width " << imageWidth  << " height " << imageHeight   std::cout << "index " << index  << " total " << total  << " errorCode " << errorcode  << " errorMag " << errorMsg  }  void test() {  auto \*loop\_ = g\_main\_loop\_new(nullptr, false);   GenAiImageSession \*session = genai\_image\_create\_session();  int initRet = genai\_image\_init\_session(*session*);  fprintf(stderr, "init return value %i\n", initRet);   ImageConfig \*config = image\_config\_create();  image\_config\_set\_generation\_number(*config*, 1);  image\_config\_set\_style(*config*, "写实风格");  image\_config\_set\_size(*config*, ImageSize{1280, 720});  genai\_image\_set\_config(*session*, *config*);   bool imageNumber = genai\_image\_get\_supported\_image\_number(*session*);    int sizeNumber;  const ImageSize\* imageSize = genai\_image\_get\_supported\_image\_size(*session*, *&sizeNumber*);  int i = 0;  while (i < sizeNumber) {  std::cout << "supported image size width " << imageSize[i].width   ++i;  }   int styleNumber;  int j = 0;  const char\*\* imageStyle = genai\_image\_get\_supported\_image\_style(*session*, *&styleNumber*);  while (j < styleNumber) {   ++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**

|  |
| --- |
| C++ 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; |