Copyright © Huawei Technologies Co., Ltd. 2020. All rights reserved.
No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Technologies Co., Ltd.

Trademarks and Permissions

and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.
1 Before You Start ....................................................................................................................... 1
  1.1 ACL Overview .................................................................................................................. 1
  1.2 Basic Concepts ............................................................................................................... 3
  1.3 Processes, Threads, Devices, Contexts, and Streams ....................................................... 5
  1.4 Memory Allocation ........................................................................................................ 8
  1.5 Sample Obtaining .......................................................................................................... 12
  1.6 Log Viewing .................................................................................................................... 14

2 API Calling Flow .................................................................................................................... 15
  2.1 Typical API Calls ......................................................................................................... 16
  2.2 Runtime Resource Allocation ....................................................................................... 17
  2.3 Model Loading ............................................................................................................. 19
  2.4 Data Preprocessing ....................................................................................................... 20
    2.4.1 JPEG Image Decoding .......................................................................................... 21
    2.4.2 JPEG Image Encoding ......................................................................................... 23
    2.4.3 Image Cropping and Resizing ............................................................................. 25
    2.4.4 Video Decoding .................................................................................................. 28
    2.4.5 Video Encoding .................................................................................................. 31
  2.5 Model Inference ............................................................................................................ 33
    2.5.1 Typical Model Inference Flow ............................................................................. 34
    2.5.2 Setting Dynamic Batch/Image Size and Dynamic AIPP ........................................ 35
    2.5.3 Preparing Input and Output Data for Model Inference ........................................ 38
  2.6 Operator Calling .......................................................................................................... 40
  2.7 Synchronous Wait ........................................................................................................ 43
    2.7.1 Multi-Device Scenario ......................................................................................... 44
    2.7.2 Multi-Stream Scenario ......................................................................................... 45
    2.7.3 Scenarios with Callback ....................................................................................... 46
  2.8 Runtime Resource Release ............................................................................................ 48
  2.9 APIs and Data Structures ............................................................................................. 49

3 Development Flow ............................................................................................................... 50

4 Environment Preparation .................................................................................................... 52

5 Your First Application ......................................................................................................... 53
  5.1 Development Requirement Analysis ............................................................................. 53
5.2 Code Directory Creation.......................................................................................................................................................... 55
5.3 Model Conversion Using ATC................................................................................................................................................ 56
5.4 Application Development ...................................................................................................................................................... 60
5.4.1 Initializing Resources.......................................................................................................................................................... 60
5.4.1.1 Initializing ACL.............................................................................................................................................................. 60
5.4.1.2 Allocating Runtime Resources (Single Process, Single Thread, Single Stream)............................................... 61
5.4.1.3 Allocating Data Pre-processing Resources .................................................................................................................. 62
5.4.1.4 Allocating Model Inference Resources ..................................................................................................................... 62
5.4.1.5 Initializing Single-Operator Information ...................................................................................................................... 64
5.4.2 Copying Data to Device...................................................................................................................................................... 64
5.4.3 Pre-Processing Data (Image Decoding and Resizing)....................................................................................................... 65
5.4.4 Performing Model Inference (Batch Size = 1, Fixed Shape, Static AIPP, Single Model)........................................ 68
5.4.5 Post-Processing Data Using Single Operator and Returning Result to Host ................................................................. 69
5.4.6 Releasing Runtime Resources and Deinitializing ACL ................................................................................................. 70
5.5 Application Build and Run...................................................................................................................................................... 71

6 ACL Functions............................................................................................................................................................................. 74
6.1 Stream Management................................................................................................................................................................. 74
6.1.1 Basic Principles................................................................................................................................................................. 74
6.1.2 Single-Thread, Single-Stream Scenario ............................................................................................................................ 74
6.1.3 Multi-Thread, Multi-Stream Scenario............................................................................................................................. 75
6.2 Synchronous Wait.................................................................................................................................................................... 75
6.2.1 Basic Principles................................................................................................................................................................. 75
6.2.2 Event Synchronous Wait.................................................................................................................................................. 75
6.2.3 Intra-Stream Task Synchronous Wait ........................................................................................................................... 76
6.2.4 Inter-Stream Task Synchronous Wait ........................................................................................................................... 76
6.2.5 Device Synchronous Wait ............................................................................................................................................... 76
6.3 Data Copy.................................................................................................................................................................................. 77
6.3.1 Basic Principles................................................................................................................................................................. 77
6.3.2 Intra-Host Data Copy...................................................................................................................................................... 77
6.3.3 Intra-Device Data Copy.................................................................................................................................................. 77
6.3.4 Host-to-Device Data Copy............................................................................................................................................ 78
6.3.5 Device-to-Host Data Copy............................................................................................................................................ 78
6.4 Data Preprocessing................................................................................................................................................................. 78
6.4.1 General Description......................................................................................................................................................... 78
6.4.2 VPC..................................................................................................................................................................................... 79
6.4.2.1 Functions and Restrictions ........................................................................................................................................ 79
6.4.2.2 Cropping.................................................................................................................................................................... 89
6.4.2.3 Cropping and Pasting............................................................................................................................................... 91
6.4.2.4 Resizing...................................................................................................................................................................... 92
6.4.3 JPEGD.....................................................................................................................................................................................93
6.4.3.1 Functions and Restrictions ........................................................................................................................................ 93
6.4.3.2 Image Decoding.......................................................................................................................................................... 96
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.3.6</td>
<td>aclrtGetDeviceCount</td>
</tr>
<tr>
<td>7.4</td>
<td>Context Management</td>
</tr>
<tr>
<td>7.4.1</td>
<td>aclrtCreateContext</td>
</tr>
<tr>
<td>7.4.2</td>
<td>aclrtDestroyContext</td>
</tr>
<tr>
<td>7.4.3</td>
<td>aclrtSetCurrentContext</td>
</tr>
<tr>
<td>7.4.4</td>
<td>aclrtGetCurrentContext</td>
</tr>
<tr>
<td>7.5</td>
<td>Streams Management</td>
</tr>
<tr>
<td>7.5.1</td>
<td>aclrtCreateStream</td>
</tr>
<tr>
<td>7.5.2</td>
<td>aclrtDestroyStream</td>
</tr>
<tr>
<td>7.6</td>
<td>Synchronous Wait</td>
</tr>
<tr>
<td>7.6.1</td>
<td>aclrtCreateEvent</td>
</tr>
<tr>
<td>7.6.2</td>
<td>aclrtDestroyEvent</td>
</tr>
<tr>
<td>7.6.3</td>
<td>aclrtRecordEvent</td>
</tr>
<tr>
<td>7.6.4</td>
<td>aclrtResetEvent</td>
</tr>
<tr>
<td>7.6.5</td>
<td>aclrtQueryEvent</td>
</tr>
<tr>
<td>7.6.6</td>
<td>aclrtSynchronizeEvent</td>
</tr>
<tr>
<td>7.6.7</td>
<td>aclrtEventElapsedTime</td>
</tr>
<tr>
<td>7.6.8</td>
<td>aclrtStreamWaitEvent</td>
</tr>
<tr>
<td>7.6.9</td>
<td>aclrtSynchronizeDevice</td>
</tr>
<tr>
<td>7.6.10</td>
<td>aclrtSynchronizeStream</td>
</tr>
<tr>
<td>7.6.11</td>
<td>aclrtSubscribeReport</td>
</tr>
<tr>
<td>7.6.12</td>
<td>aclrtLaunchCallback</td>
</tr>
<tr>
<td>7.6.13</td>
<td>aclrtProcessReport</td>
</tr>
<tr>
<td>7.6.14</td>
<td>aclrtUnSubscribeReport</td>
</tr>
<tr>
<td>7.6.15</td>
<td>aclrtSetExceptionInfoCallback</td>
</tr>
<tr>
<td>7.6.16</td>
<td>aclrtGetTaskIdFromExceptionInfo</td>
</tr>
<tr>
<td>7.6.17</td>
<td>aclrtGetStreamIdFromExceptionInfo</td>
</tr>
<tr>
<td>7.6.18</td>
<td>aclrtGetThreadIdFromExceptionInfo</td>
</tr>
<tr>
<td>7.7</td>
<td>Memory Management</td>
</tr>
<tr>
<td>7.7.1</td>
<td>aclrtMalloc</td>
</tr>
<tr>
<td>7.7.2</td>
<td>aclrtFree</td>
</tr>
<tr>
<td>7.7.3</td>
<td>aclrtMallocHost</td>
</tr>
<tr>
<td>7.7.4</td>
<td>aclrtFreeHost</td>
</tr>
<tr>
<td>7.7.5</td>
<td>aclrtMemset</td>
</tr>
<tr>
<td>7.7.6</td>
<td>aclrtMemsetAsync</td>
</tr>
<tr>
<td>7.7.7</td>
<td>aclrtMemcpy</td>
</tr>
<tr>
<td>7.7.8</td>
<td>aclrtMemcpyAsync</td>
</tr>
<tr>
<td>7.8</td>
<td>Model Loading and Execution</td>
</tr>
<tr>
<td>7.8.1</td>
<td>aclmdlLoadFromFile</td>
</tr>
<tr>
<td>7.8.2</td>
<td>aclmdlLoadFromMem</td>
</tr>
<tr>
<td>7.8.3</td>
<td>aclmdlLoadFromFileWithMem</td>
</tr>
<tr>
<td>7.8.4</td>
<td>aclmdlLoadFromMemWithMem</td>
</tr>
</tbody>
</table>
7.15.20.1 aclCreateDataBuffer.................................................................................................................................272
7.15.20.2 aclDestroyDataBuffer.................................................................................................................................273
7.15.20.3 aclGetDataBufferAddr.................................................................................................................................273
7.15.20.4 aclGetDataBufferSize..................................................................................................................................274
7.15.21 aclmdlDataset......................................................................................................................................................274
7.15.21.1 aclmdlCreateDataset...................................................................................................................................275
7.15.21.2 aclmdlDestroyDataset...................................................................................................................................275
7.15.21.3 aclmdlAddDatasetBuffer...............................................................................................................................275
7.15.21.4 aclmdlGetDatasetNumBuffers.......................................................................................................................276
7.15.21.5 aclmdlGetDatasetBuffer................................................................................................................................276
7.15.22 aclmdlDesc.........................................................................................................................................................277
7.15.22.1 aclmdlCreateDesc.........................................................................................................................................277
7.15.22.2 aclmdlDestroyDesc.........................................................................................................................................277
7.15.22.3 aclmdlGetDesc...............................................................................................................................................278
7.15.22.4 aclmdlGetNumInputs.....................................................................................................................................278
7.15.22.5 aclmdlGetNumOutputs..................................................................................................................................279
7.15.22.6 aclmdlGetInputSizeByIndex..........................................................................................................................279
7.15.22.7 aclmdlGetOutputSizeByIndex.........................................................................................................................280
7.15.22.8 aclmdlGetInputDims.....................................................................................................................................280
7.15.22.9 aclmdlGetOutputDims...................................................................................................................................281
7.15.22.10 aclmdlGetInputNameByIndex.......................................................................................................................282
7.15.22.11 aclmdlGetOutputNameByIndex...................................................................................................................282
7.15.22.12 aclmdlGetInputFormat..................................................................................................................................283
7.15.22.13 aclmdlGetOutputFormat................................................................................................................................283
7.15.22.14 aclmdlGetInputDataType............................................................................................................................284
7.15.22.15 aclmdlGetOutputDataType............................................................................................................................285
7.15.22.16 aclmdlGetInputIndexByName.......................................................................................................................285
7.15.22.17 aclmdlGetOutputIndexByName....................................................................................................................286
7.15.22.18 aclmdlGetDynamicBatch.............................................................................................................................286
7.15.22.19 aclmdlGetDynamicHW.................................................................................................................................287
7.15.22.20 aclmdlGetCurOutputDims..............................................................................................................................288
7.15.22.21 aclmdlGetFirstApplInfo...............................................................................................................................288
7.15.23 aclTensorDesc......................................................................................................................................................289
7.15.23.1 aclCreateTensorDesc....................................................................................................................................289
7.15.23.2 aclDestroyTensorDesc...................................................................................................................................290
7.15.23.3 aclGetTensorDescType....................................................................................................................................291
7.15.23.4 aclGetTensorDescFormat...............................................................................................................................291
7.15.23.5 aclGetTensorDescSize........................................................................................................................................291
7.15.23.6 aclGetTensorDescElementCount.................................................................292
7.15.23.7 aclGetTensorDescNumDims........................................................................292
7.15.23.8 aclGetTensorDescDim.................................................................................293
7.15.23.9 aclSetTensorDescName.................................................................................293
7.15.23.10 aclGetTensorDescName.............................................................................294
7.15.23.11 aclTransTensorDescFormat.................................................................294
7.15.23.12 aclSetTensorStorageShape.................................................................295
7.15.23.13 aclSetTensorStorageFormat.................................................................295
7.15.24 aclopAttr........................................................................................................296
7.15.24.1 aclopCreateAttr.........................................................................................296
7.15.24.2 aclopDestroyAttr.......................................................................................296
7.15.24.3 aclopSetAttrBool.....................................................................................297
7.15.24.4 aclopSetAttrInt.......................................................................................297
7.15.24.5 aclopSetAttrFloat....................................................................................298
7.15.24.6 aclopSetAttrString...................................................................................298
7.15.24.7 aclopSetAttrListBool..............................................................................299
7.15.24.8 aclopSetAttrListInt...............................................................................299
7.15.24.9 aclopSetAttrListFloat.............................................................................300
7.15.24.10 aclopSetAttrListString......................................................................301
7.15.24.11 aclopSetAttrListListInt......................................................................301
7.15.25 acldvppChannelDesc.....................................................................................302
7.15.25.1 acldvppCreateChannelDesc...................................................................302
7.15.25.2 acldvppDestroyChannelDesc.................................................................302
7.15.25.3 acldvppGetChannelDescChannelId.......................................................303
7.15.25.4 acldvppSetChannelDescMode.................................................................304
7.15.26 acldvppPicDesc..........................................................................................304
7.15.26.1 acldvppCreatePicDesc..........................................................................304
7.15.26.2 acldvppSetPicDesc Series.....................................................................305
7.15.26.3 acldvppGetPicDesc Series....................................................................306
7.15.26.4 acldvppDestroyPicDesc.........................................................................307
7.15.27 acldvppRoiConfig.......................................................................................307
7.15.27.1 acldvppCreateRoiConfig.......................................................................307
7.15.27.2 acldvppSetRoiConfig Series...................................................................308
7.15.27.3 acldvppDestroyRoiConfig......................................................................309
7.15.28 acldvppResizeConfig..................................................................................310
7.15.28.1 acldvppCreateResizeConfig..................................................................310
7.15.28.2 acldvppSetResizeConfigInterpolation..................................................310
7.15.28.3 acldvppGetResizeConfigInterpolation..................................................311
7.15.28.4 acldvppDestroyResizeConfig..................................................................311
7.15.29 acldvppJpegeConfig....................................................................................312
7.15.29.1 acldvppCreateJpegeConfig....................................................................312
8 ACL Sample Description

8.1 Samples................................................................................................................................................. 336
8.2 Matrix-matrix multiplication.................................................................................................................. 338
8.2.1 Sample Overview................................................................................................................................. 338
8.2.2 Building and Running Application on Host...................................................................................... 340
8.3 Image Classification based on Caffe ResNet-50 (Including Image Decoding and Resizing)............. 342
8.3.1 Sample Overview................................................................................................................................... 342
8.3.2 Building and Running Application on Host...................................................................................... 345
8.4 Image Classification based on Caffe ResNet-50 (Including Image Decoding, Cropping, Resizing, and Encoding)............................................................................................................. 348
8.4.1 Sample Overview.................................................................................................................................... 348
8.4.2 Building and Running Application on Host...................................................................................... 351
8.5 Image Classification Based on Caffe ResNet-50 (Including Video Decoding)................................... 356
8.5.1 Sample Overview................................................................................................................................... 356
8.5.2 Building and Running Application on Host...................................................................................... 358
8.6 Image Classification Based on Caffe ResNet-50 (Synchronous Inference, Excluding Data Pre-Processing)........................................................................................................................................ 362
8.6.1 Sample Overview................................................................................................................................... 362
8.6.2 Building and Running Application on Host...................................................................................... 365
8.7 Image Classification Based on Caffe ResNet-50 (Asynchronous Inference, Excluding Data Pre-Processing)........................................................................................................................................ 368
8.7.1 Sample Overview................................................................................................................................... 368
8.7.2 Building and Running Application on Host...................................................................................... 371
8.8 CompileDemo........................................................................................................................................... 375
8.8.1 Sample Principles.................................................................................................................................... 375
8.8.2 Code Implementation Parsing.............................................................................................................. 376
8.8.3 Code Directory....................................................................................................................................... 377
8.8.4 Code Compilation................................................................................................................................... 378
8.8.4.1 Initializing ACL................................................................................................................................. 378
8.8.4.2 Runtime Resource Allocation...................................................................................................... 378
8.8.4.3 "Hello DaVinci" Transmission and Backhaul.............................................................................. 379
8.8.4.4 Runtime Resource Release...................................................................................................... 379
8.8.5 Code Compilation and Running........................................................................................................ 380
8.9 InferClassification................................................................................................................................... 385
8.9.1 Sample Principles................................................................................................................................... 385
8.9.2 Code Implementation Parsing.............................................................................................................. 387
8.9.3 Code Directory....................................................................................................................................... 400
8.9.4 Code Compilation................................................................................................................................... 402
8.9.4.1 Main Program....................................................................................................................................... 402
8.9.4.2 Resource Initialization....................................................................................................................... 405
1 Before You Start

This document provides guidance for developers to develop deep neural network (DNN) applications for purposes including object detection and image classification based on existing models and the C++ APIs provided by the Ascend Computing Language (ACL).

1.1 ACL Overview

1.2 Basic Concepts

1.3 Processes, Threads, Devices, Contexts, and Streams

1.4 Memory Allocation

1.5 Sample Obtaining

1.6 Log Viewing

1.1 ACL Overview

Ascend Computing Language (ACL) provides a collection of C++ APIs for users to develop deep neural network applications for object recognition and image classification, ranging from device management, context management, streams management, memory management, to model loading and execution, operator loading and execution, and media data processing. Users can call ACL APIs directly or through a third-party framework to utilize the computing capability of the Ascend AI Processor, or encapsulate the ACL APIs to implement a third-party library, to utilize the execution and resources management capabilities of the Ascend AI Processor.

When running an application, ACL calls APIs provided by the Graph Engine Executor (GE) to load and run models and operators, and calls Runtime APIs to manage devices, contexts, streams, and memory.

The AI Core and AI CPU, as the hardware computing capability basis of the Ascend AI Processor, perform matrix-related computation of neural networks, general computation and execution control of control operators, scalars, and vectors, as well as image and video data preprocessing. AI Core and AI CPU guarantee the execution of the deep neural network (NN) computing.
**Figure 1-1 Logical architecture**

- **Apps**
  - Third-party frameworks
  - Third-party libraries

- **Ascend Computing Language (ACL)**
  - Model loading
  - Open operator capabilities
  - Open Runtime

- **Graph Engine (GE) Executor**

- **Runtime**

- **Driver**

- **AI Core/AI CPU (computing resources)**

- **Software stack**
1.2 Basic Concepts

Table 1-1 Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous/Asynchronous</td>
<td>Scheduling is classified into the synchronous and asynchronous modes in this document from the perspective of the caller and executor. In the current scenario, if the API calling result is returned immediately without waiting for the device to complete the execution, the host scheduling is in asynchronous mode. If the API calling result is not returned until the device completes the execution, the host scheduling is in synchronous mode.</td>
</tr>
<tr>
<td>Process/Thread</td>
<td>Unless otherwise specified, the processes and threads mentioned in this document refer to the processes and threads on the host.</td>
</tr>
<tr>
<td>Host</td>
<td>The host side refers to the x86 or Arm server connected to the device side. The host side utilizes the NN computing capability provided by the device side to implement services.</td>
</tr>
<tr>
<td>Device</td>
<td>The device side refers to the hardware device that provides the host with the NN computing capability over the PCIe interface.</td>
</tr>
</tbody>
</table>
| Context               | As a container, the context manages the life cycle its objects (including streams, events, and device memory). Streams and events in different contexts are isolated, and cannot be executed synchronously. There are two types of contexts:  
  - Default context: a default context created implicitly upon the aclrtSetDevice call that specifies the device for computation. Each device corresponds to a default context. A default context cannot be released by calling aclrtDestroyContext.  
  - Explicitly created context (recommended): a context created explicitly upon the aclrtCreateContext call in a process or thread |
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
</table>
| Stream | Streams preserve the execution order of a stack of asynchronous operations that are executed on the device in its original order. Stream-based kernel execution and data transfer can implement the parallelism of computing on both the host and device sides, and data transfer between the host and device sides. Streams come in two types:  
  - Default stream: a default stream created implicitly upon the `aclrtSetDevice` call that specifies the device for computation. Each device corresponds to a default stream. A default stream cannot be released by calling `aclrtDestroyStream`.  
  - Explicitly created stream (recommended): a stream created explicitly upon the `aclrtCreateStream` call in a process or thread. |
| Event | Supports tasks synchronization between streams by calling ACL APIs, including tasks between the host and device, and tasks between the devices. For example, if task of stream2 needs to be executed after task in stream1 is complete, you can create an event and insert it to stream1. |
| AIPP | The AIPP module is introduced for image pre-processing including Color Space Conversion (CSC), image normalization (by subtracting the mean value or multiplying a factor), image cropping (by specifying the crop start and cropping the image to the size required by the neural network), and much more. AIPP supports static and dynamic modes. However, the two modes are mutually exclusive.  
  - Static AIPP: During model conversion, set the AIPP mode to static and set the AIPP parameters. After the model is generated, the AIPP parameter values are saved in the offline model (*.om). The same AIPP parameter configurations are used in each model inference phase. In static AIPP mode, batches share the same set of AIPP parameters.  
  - Dynamic AIPP: During model conversion, specify the AIPP mode to dynamic, and set different sets of dynamic AIPP parameters as required. In this way, different sets of parameters can be used for model inference. For details about the interface for setting dynamic AIPP parameters, see 7.15.17.2 Dynamic AIPP Parameters Setting. In dynamic AIPP mode, batches can use different sets of AIPP parameters. |
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
</table>
| Dynamic batch/image size    | In some scenarios, the batch size or image size is not fixed. For example, if a facial recognition network is executed after face detection, the number of faces inferred each time is not fixed.  
  - Dynamic batch: The batch size is not fixed during inference.  
  - Dynamic image size: The image size (H x W) of each image is not fixed during inference.                                                                                                                                   |
| Channel                     | An RGB image has three channels: red, green, and blue. HSV stands for hue, saturation, and value (brightness) and is an alternative representation of the RGB color model.                                                                                                                    |

### 1.3 Processes, Threads, Devices, Contexts, and Streams

For details about the terms, see [1.2 Basic Concepts](#).

**Relationships Between Devices, Contexts, and Streams**

- A context must belong to one device.  
- Contexts can be created implicitly or explicitly.  
- The life cycle of an implicitly created context starts with the `aclrtSetDevice` call and ends with the `aclrtDestroyContext` call when the reference count is 0. An implicitly created context is created only once. Calling `aclrtSetDevice` repeatedly to specify the same device adds only the reference count of the implicitly created context.  
- The life cycle of an explicitly created context starts with the `aclrtCreateContext` call and ends with the `aclrtDestroyContext` call.  
- A context is bound to a user thread. One context is used in a thread at a time. The context can be set by calling `aclrtSetCurrentContext`. An implicitly created context does not need to be specified by calling `aclrtSetCurrentContext`.  
- Contexts in a process can be switched by calling `aclrtSetCurrentContext`.  

![Diagram](image)

**Relationships Between Threads, Contexts, and Streams**

- A context must be bound to a user thread. The usage and scheduling of all device resources must be based on the context.  
- Only one context that is associated with the device is used in a thread at a time.  
- `aclrtSetCurrentContext` can be called to quickly switch between devices. A code sample is provided as follows.

```c
...  
aclrtCreateContext(&ctx1,0);  
aclrtCreateStream(&s1);  
aclopExecute(op1,...,s1);  
aclrtCreateContext(&ctx2,1);  
/* After ctx1 is created, the context used in the current thread changes to ctx2, and the corresponding tasks are computed on device 1. In this sample, op2 is executed on device 1. */
```
Multiple streams can be created in a thread, where tasks in different streams can be implemented in parallel. In multi-thread scenarios, you can also create one stream in each thread, where each stream is independent on the device, and tasks in each stream are executed in its original order.

Multi-thread scheduling depends on the OS scheduling of the running application. Multi-stream scheduling is performed by the scheduling component on the device side.

**Context Migration Between Threads in a Process**

- Multiple contexts can be created in a process, but only one context is used in a thread at a time.
- If multiple contexts are created in a thread, the last created context is used by default.
- If multiple contexts are created in a process, call `aclrtSetCurrentContext` to set the context to be used.
Application Scenarios of Default Contexts and Default Streams

- Before operations are delivered on the device, a context and a stream must exist. Implicitly created contexts and streams are default contexts and streams.
- If an implicitly created context is used, `aclrtGetCurrentContext`, `aclrtSetCurrentContext` or `aclrtDestroyContext` is not available.
- Implicitly created contexts and streams are applicable to simple applications where only one device is needed for computation. For multi-thread
applications, you are advised to use the explicitly created contexts and streams.

A code sample is provided as follows.

```c
... aclInit(...); aclrtSetDevice(0);
/* In the default context, a default stream is created, and the stream is available in the current thread. */
... aclopExecute(op1,...,NULL); //NULL indicates that op1 is executed in the default stream.
aclopExecute(op2,...,NULL); //NULL indicates that op2 is executed in the default stream.
aclrtSynchronizeStream(NULL);
/* Output the result as required when all computing tasks (op1 and op2 execution tasks) are complete. */
... aclrtResetDevice(0); // Reset device 0. The life cycles of the corresponding default context and default stream end.
```

### Multi-Thread and Multi-Stream Performance

- Thread scheduling depends on the OS. The device scheduling unit schedules tasks in streams. When tasks in different streams in a process contend for resources on the device, the performance may be lower than that of a single stream scenario.

- The processor provides different execution components, such as the AI Core, AI CPU and Vector Core, so that different tasks can be executed by different components. You are advised to create streams based on the operator execution engines.

- The performance depends on the logic implementation of the application. Generally, the performance of a single-thread, multi-stream scenario is slightly better than that of a multi-thread, single-stream scenario, because less thread scheduling is involved at the application layer.

### 1.4 Memory Allocation

Memory can be allocated in either of the following ways:

1. Allocate memory independently as required without split or re-allocation.
2. Allocate a large memory pool at a time and re-allocate the memory from the memory pool as required.

Call the following APIs for memory re-allocation. Pay attention to the restrictions on the memory address and memory size of each API. Otherwise, memory overwriting may occur.

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
<th>Input/Output Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>aclrtMemcpyAsync</td>
<td>Implements asynchronous memory copy between the host and device.</td>
<td>For on-chip memory copy within the device, the source and destination address must be 64-byte aligned.</td>
</tr>
</tbody>
</table>
### API Description

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
<th>Input/Output Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>aclrtMalloc</td>
<td>Allocates memory on the device. It is a synchronous interface.</td>
<td>• If the memory is allocated by using the <code>aclrtMalloc</code> interface, the memory needs to be released by calling <code>aclrtFree</code>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Performance deterioration will be caused by the frequent calling of <code>aclrtMalloc</code> to allocate memory and <code>aclrtFree</code> to free memory. You are advised to allocate or manage memory in advance to avoid frequent memory application and freeing.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The allocated memory size is the input size rounded up to the nearest multiple of 32, plus 32 bytes.</td>
</tr>
<tr>
<td>acldvppMalloc</td>
<td>Allocates memory for media processing on the device. The allocated huge memory page meets the data processing requirements (for example, the start address is 128-byte aligned). It is a synchronous interface. <code>acldvppMalloc</code> and <code>acldvppFree</code> must be used in pair. Memory allocated using <code>acldvppMalloc</code> can only be freed by calling <code>acldvppFree</code>. For details about the media data processing, see 7.12 Media Data Processing.</td>
<td>When the VPC function is implemented, the requirements for the input and output memory are as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The start address must be 16-byte aligned. Call <code>acldvppMalloc</code> or <code>acldvppFree</code> to allocate, or free memory respectively.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The memory size is related to the image data format, and the calculation formulas are as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>YUV400SP and YUV420SP: <code>widthStride x heightStride x 3/2</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>YUV422SP: <code>widthStride x heightStride x 2</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>YUV444SP: <code>widthStride x heightStride x 3</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>YUV442 packed: <code>widthStride x heightStride</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>YUV444 packed and RGB888: <code>widthStride x heightStride</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>XRGB8888: <code>widthStride x heightStride</code></td>
</tr>
<tr>
<td>API</td>
<td>Description</td>
<td>Input/Output Memory</td>
</tr>
<tr>
<td>-----</td>
<td>-------------</td>
<td>---------------------</td>
</tr>
<tr>
<td></td>
<td>When the JPEGD function is implemented:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Input memory:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- The size of the input memory is the size of the input image.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- The start address of the input memory must be 128-byte aligned. Allocate device memory by calling <code>acldevppMalloc</code> and free device memory by calling <code>acldevppFree</code>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Output memory:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Calculate the output memory size according to the image format.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>YUV420 SP: widthStride x heightStride x 3/2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>YUV422 SP: widthStride x heightStride x 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>YUV444 SP: widthStride x heightStride x 3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- The start address of the output memory must be 128-byte aligned. Allocate device memory by calling <code>acldevppMalloc</code> and free device memory by calling <code>acldevppFree</code>. For huge memory: Memory size = Output memory size + (n – 1) x AlignTo128(Output memory size + 8), where, n indicates the number of images.</td>
<td></td>
</tr>
<tr>
<td>API</td>
<td>Description</td>
<td>Input/Output Memory</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>---------------------</td>
</tr>
<tr>
<td></td>
<td>When the JPEG Encode function is implemented:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Input memory:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Calculate the input memory size according to the image format.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>YUV422 Packed: \texttt{widthStride} \times \texttt{heightStride}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>YUV420 SP: \texttt{widthStride} \times \texttt{heightStride} \times 3/2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The start address of the input memory must be 128-byte aligned. Allocate device memory by calling \texttt{acldevpMalloc} and free device memory by calling \texttt{acldevpFree}.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Output memory:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The size of the output memory is the size of the encoded image.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The start address of the output memory must be 128-byte aligned. Allocate device memory by calling \texttt{acldevpMalloc} and free device memory by calling \texttt{acldevpFree}.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>When the Video Decode function is implemented:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Input memory:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Allocate device memory by calling \texttt{aclrtMalloc} and free memory by calling \texttt{aclrtFree}. Alternatively, allocate device memory by calling \texttt{acldevpMalloc} and free memory by calling \texttt{acldevpFree}.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Output memory:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Calculate the output memory size according to the image format.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>YUV420 SP: \texttt{widthStride} \times \texttt{heightStride} \times 3/2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The start address of the output memory must be 16-byte aligned. Allocate device memory by calling \texttt{aclrtMalloc} and free device memory by calling \texttt{aclrtFree}.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>When the Video Encode function is implemented:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Input memory:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Allocate device memory by calling \texttt{acldevpMalloc} and free device memory by calling \texttt{acldevpFree}.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Output memory:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The output memory is managed by the system.</td>
<td></td>
</tr>
</tbody>
</table>
### API Description

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
<th>Input/Output Memory</th>
</tr>
</thead>
</table>
| aclrtMallocHost | Allocates memory on the host. It is a synchronous interface.               | • The allocated memory cannot be used on the device and needs to be explicitly copied to the device.  
• If the memory is allocated by using the aclrtMallocHost interface, the memory needs to be released by calling aclrtFreeHost. |

The typical memory management scenarios during media data processing are as follows:

**Figure 1-2 VDEC scenario**

![VDEC scenario diagram]

**Figure 1-3 JPEGD scenario**

![JPEGD scenario diagram]

### 1.5 Sample Obtaining

API calling code samples are provided for you to get started.
Table 1-2 describes the important functions of each sample. For details about directory, basic principles, and usage of each sample, see Application Development > Application Build and Run in Application Software Development Guide.

- **acl_execute_gemm**: implements matrix-matrix multiplication.
- **acl_dvpp_resnet50**: classifies images using Caffe ResNet-50 (single input and single batch), including image data preprocessing (decoding and resizing).
- **acl_vpc_jpege_resnet50**: classifies images using Caffe ResNet-50 (single input and single batch), including image decoding, cropping, resizing, and encoding.
- **acl_vdec_resnet50**: classifies images using Caffe ResNet-50 (single input and single batch), including video decoding and encoding.
- **acl_resnet50**: classifies images using Caffe ResNet-50 (single input and single batch), excluding image data preprocessing (decoding and resizing). It is synchronous inference.
- **acl_resnet50_async**: classifies images using Caffe ResNet-50 (single input and single batch), excluding image data preprocessing (decoding and resizing). It is asynchronous inference.
- **dvpp**: implements image cropping, resizing, decoding, and encoding, as well as video decoding and encoding with Digital Vision Preprocessing (DVPP).

<table>
<thead>
<tr>
<th>Sample Name</th>
<th>Device Management</th>
<th>Context Management</th>
<th>Stream Management</th>
<th>Synchronous Wait</th>
<th>Data Copy</th>
<th>Data Preprocessing</th>
<th>Model Inference</th>
<th>Data Postprocessing</th>
<th>Single-Operator Calling</th>
</tr>
</thead>
<tbody>
<tr>
<td>acl_execute_gemm</td>
<td>-</td>
<td>-</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>√</td>
</tr>
<tr>
<td>acl_dvpp_resnet50</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>acl_vpc_jpege_resnet50</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>acl_vdec_resnet50</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>acl_resnet50</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>-</td>
<td>-</td>
<td>√</td>
<td>√</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
## 1.6 Log Viewing

If an error occurs during application running, obtain the log file by referring to Log Reference and check out the error details.
2 API Calling Flow

2.1 Typical API Calls
2.2 Runtime Resource Allocation
2.3 Model Loading
2.4 Data Preprocessing
2.5 Model Inference
2.6 Operator Calling
2.7 Synchronous Wait
2.8 Runtime Resource Release
2.9 APIs and Data Structures
2.1 Typical API Calls

Figure 2-1 API calling flowchart

The API calling flowchart describes the typical API calls. If the size of the input image does not meet the model requirements, data preprocessing is required. To implement model inference, the model needs to be loaded first. After the completion of model inference, the model needs to be unloaded. After model inference, the inference result needs to be postprocessed for looking up the class with the highest confidence score.

1. ACL initialization: Call aclInit to initialize the ACL.
2. Runtime resource allocation: Allocate devices, contexts and streams in sequence. For details, see 2.2 Runtime Resource Allocation.
3. Operator calling. For details, see 2.6 Operator Calling.
a. Operator .om file generation: Build the .json definition file into an offline model supported by the Ascend AI processor (.om file) with the ATC. For details, see ATC Tool Instructions.
b. Operator .om file loading for operator running
c. Operator execution for result output

4. Model inference
a. Model .om file generation: Convert the third-party network (for example, Caffe ResNet-50) into an offline model adapted to the Ascend AI Processor (.om file) with the ATC. For details, see ATC Tool Instructions.
b. Model loading: Load the model to the system for model inference. For details, see 2.3 Model Loading.
c. (Optional) Data preprocessing: Implement JPEG image decoding, video decoding, image cropping, resizing and format conversion, as well as JPEG image encoding. For details, see 2.4 Data Preprocessing.
d. Model inference: Implement object recognition and image classification with a model. Currently, the ACL provides synchronous and asynchronous inference APIs, supporting dynamic batch/image size, and dynamic AIPP scenarios. For details, see 2.5 Model Inference.
e. (Optional) Data postprocessing: Postprocess the model inference result as required. For example, you can export the inference result to a file, and look up in the inference result for the class with the highest confidence score using single operators.
f. Model unloading: Call aclmdlUnload to unload a model.

5. Runtime resource release: Release streams, contexts and devices in sequence. For details, see 2.8 Runtime Resource Release.

6. ACL deinitialization: Call aclFinalize to deinitialize the ACL.

**NOTE**

Memory allocation and release, data transfer (by memory copy), and data type creation and destruction are the most frequently-used calls in each phase of the application development, and therefore are not thoroughly illustrated in the flowchart. For details about memory allocation and release, and memory copy APIs, see 7.7 Memory Management. For details about data type creation and destruction APIs, see 7.15 Data Types and Operation APIs.

### 2.2 Runtime Resource Allocation

You need to allocate runtime resources devices, contexts and streams in sequence. Contexts and streams can be created implicitly or explicitly.

- Implicitly created contexts and streams: They are applicable to simple applications without complex interaction logic. However, in multi-thread programming, the execution result depends on the thread scheduling sequence.
- Explicitly created contexts and streams (recommended): They are applicable to large-scale applications with complex interaction logic, offering better application readability and maintainability.
Figure 2-2 Runtime resource allocation flowchart

1. Allocate devices, contexts and streams in sequence.
   - Call aclrtSetDevice to explicitly specify a device for computation.
     - Call aclrtCreateContext to explicitly create a context. Call aclrtCreateStream to explicitly create a stream.
     - If the contexts and streams are not created explicitly, the system uses the default contexts and streams, which are created implicitly with the aclrtSetDevice call.
       To pass the default stream to any API call, pass NULL directly.
   - Implicitly specify a device for computation. Call aclrtCreateContext to explicitly create a context. Call aclrtCreateStream to explicitly create a stream. When explicitly creating
the context, a device for computation is specified by the system with the \texttt{aclrtSetDevice} call. The \texttt{deviceId} is passed to the \texttt{aclrtCreateContext} call.

2. (Optional) If the executable file of an application can be executed on both the host and device, call \texttt{aclrtGetRunMode} to obtain the running mode of the software stack during programming, for determining the calling logic of the memory allocation APIs.

   a. If the executable file of an application is executed on the host, data transfer between the host and the device may be involved. In this case, the \texttt{aclrtMemcpy} interface (synchronous mode) or \texttt{aclrtMemcpyAsync} interface (asynchronous mode) needs to be called to implement data transmission through memory copy.

   b. If the executable file of the application is executed on the device, data transfer between the host and the device is not involved.

2.3 Model Loading

\textbf{Figure 2-3} Model loading flowchart

- Before model loading, convert the third-party network (for example, Caffe ResNet-50) into an offline model adapted to the Ascend AI Processor (.om file) with the ATC. For details, see ATC Tool Instructions.
  - In fixed-batch scenarios with \texttt{batchSize} > 1, set \texttt{batchSize} in the \texttt{input_shape} parameter during model conversion.
  - In dynamic batch size scenarios, set the \texttt{batchSize} choices in the \texttt{dynamic_batch_size} parameter during model conversion.
  - In dynamic image size scenarios, set the \texttt{dynamic_image_size} choices for model conversion.
- In dynamic AIPP scenarios, set `aipp_mode` to `dynamic` in the configuration file of the `insert_op_conf` parameter during model conversion.

- Model data can be loaded using the following APIs. A model ID is returned after the model is successfully loaded.
  - `aclmdlLoadFromFile`: loads offline model data from a file. The memory is managed by the system.
  - `aclmdlLoadFromMem`: loads offline model data from the memory. The memory is managed by the system.
  - `aclmdlLoadFromFileWithMem`: loads offline model data from a file. The memory (including working memory for storing input and output data of the model, and weight memory for storing weight data) is managed by the user.
  - `aclmdlLoadFromMemWithMem`: loads offline model data from the memory. The memory (including working memory and weight memory) is managed by the user.

### 2.4 Data Preprocessing

Data preprocessing implements JPEG image and video decoding, image cropping and resizing, as well as JPEG image and video encoding.
2.4.1 JPEG Image Decoding

The system supports .jpg, .jpeg, .JPG, and .JPEG image decoding, and outputs YUV420SP, YUV422SP, and YUV444SP images based on the source image encoding format. The major steps are as follows:

1. Call `aclDvppCreateChannel` to create a channel for image data processing.
Before creating a channel, call `acldvppCreateChannelDesc` to create the channel description.

2. Before decoding JPEG images, call `acldvppMalloc` to allocate memory on the device for storing the input or output data.
   Before allocating output memory, call `acldvppJpegPredictDecSize` to predict the output memory size required for the decoded JPEG image based on the memory used for storing JPEG image data on the host.

3. Call the asynchronous API `acldvppJpegDecodeAsync` for decoding.
   For asynchronous decoding, `aclrtSynchronizeStream` needs to be called to block the host until all tasks in the specified stream are complete.

4. After the encoding is complete, call `acldvppFree` to free the input and output memory in a timely manner.

5. Call `acldvppDestroyChannel` to destroy the image data processing channel.
   Call `acldvppDestroyChannelDesc` to destroy the channel description after the channel is destroyed.
2.4.2 JPEG Image Encoding

**Figure 2-5** JPEG image encoding flowchart

The system supports YUV to JPG image encoding. The major steps are as follows:

1. Call `aclDvppCreateChannel` to create a channel for image data processing. Before creating a channel, call `aclDvppCreateChannelDesc` to create the channel description.
2. Call `acldvppCreateJpegeConfig` to create image encoding configuration data.

3. Before decoding JPEG images, call `acldvppMalloc` to allocate memory on the device for storing the input or output data. Before allocating output memory, you can call `acldvppJpegPredictEncSize` to predict the output memory size required for the encoded image based on the input image description and the image encoding configuration data.

4. Call the asynchronous API `acldvppJpegEncodeAsync` for encoding. For asynchronous decoding, `aclrtSynchronizeStream` needs to be called to block the host until all tasks in the specified stream are complete.

5. Call `acldvppDestroyJpegeConfig` to destroy the image encoding configuration data.

6. After the encoding is complete, call `acldvppFree` to free the input and output memory in a timely manner.

7. Call `acldvppDestroyChannel` to destroy the image data processing channel. Call `acldvppDestroyChannelDesc` to destroy the channel description after the channel is destroyed.
2.4.3 Image Cropping and Resizing
Figure 2-6 Cropping and resizing flowchart

1. Allocate memory for storing input image: addbppMalloc
2. Create input image description: addbppCreatePicDesc
3. Set input image description attributes: addbppSetPicDesc
4. Create cropArea position configuration: addbppCreateRoIConfig
5. Create resizing configuration: addbppCreateResizeConfig
6. Input preparation
7. Allocate memory for storing output image: addbppMalloc
8. Create the output image description: addbppCreatePicDesc
9. Set output image description attributes: addbppSetPicDesc
10. Perform asynchronous cropping: addbppPspCropAsync
11. Set resizing algorithm: addbppSetResizeConfigInterpolation
12. Perform asynchronous resizing: addbppPspResizeAsync
13. Deliver asynchronous tasks to the same stream
14. Perform synchronous wait: addbppSynchronousStream
15. Obtain cropping and resizing results for subsequent use
16. Free the memory: addbppFree
17. Set cropArea position configuration: addbppDestroyRoIConfig
18. Destroy resizing configuration: addbppDestroyResizeConfig
19. Destroy the channel: addbppDestroyChannel
20. Destroy the channel description: addbppDestroyChannelDesc

Required
Optional
The system supports image cropping and resizing. The major steps are as follows:

1. Call `aclDvppCreateChannel` to create a channel for image data processing. Before creating a channel, call `aclDvppCreateChannelDesc` to create the channel description.

2. Call `aclDvppCreateRoiConfig` and `aclDvppCreateResizeConfig` to create `cropArea` position configuration and resizing configuration, respectively.

3. Before cropping and resizing, call `aclDvppMalloc` to allocate memory on the device for storing the input or output data.

4. Perform cropping and resizing.
   - About cropping:
     - Call the asynchronous API `aclDvppVpcCropAsync` to crop the image according to `cropArea` and loads the cropped image to the output memory for output.
     - Resize the cropped image again if the size is different from that of the canvas.
     - The asynchronous API `aclDvppVpcCropAndPasteAsync` is provided for cropping the source image based on the specified area, and pasting the cropped image to the specified area in the target image as the destination image.
       - Resize the image again if the size of the `cropArea` is inconsistent with that of the `pasteArea`.
       - To paste the `pasteArea` on the canvas loaded to the output buffer, modify the code logic as follows: allocate the output buffer and then load a canvas to the allocated buffer.
   - About resizing:
     - Call the `aclDvppVpcResizeAsync` asynchronous API to resize the input image to the size of the output image.
     - Calculate the buffer for storing the resized image based on the YUV420SP format as follows: `widthStride x heightStride x 3/2`.
     - For asynchronous decoding, `aclrtSynchronizeStream` needs to be called to block the host until all tasks in the specified stream are complete.

5. Call `aclDvppFree` to free the input and output memory.

6. Call `aclDvppDestroyRoiConfig` and `aclDvppDestroyResizeConfig` to destroy the `cropArea` position configuration and resize configuration, respectively.

7. Call `aclDvppDestroyChannel` to destroy the image data processing channel.
   - Call `aclDvppDestroyChannelDesc` to destroy the channel description after the channel is destroyed.
2.4.4 Video Decoding

Figure 2-7 Video decoding flowchart

The user thread tid1 and callback thread tid2 must be created in advance.
The major steps are as follows:

1. Call **aclvdecCreateChannel** to create a channel for video stream data processing.
   - Perform the following steps before creating a channel for processing video stream data.
     i. Call **aclvdecCreateChannelDesc** to create channel description.
     ii. Call **aclvdecSetChannelDesc series** to set the decoding channel description attributes, including the decoding channel ID, thread, callback function, and video encoding protocol.

       1) The callback function needs to be created by the user in advance. It is used to obtain the decoding data after video decoding and free resources in a timely manner. For details about the callback function prototype, see 7.12.7.3 **aclvdecCallback**.

       Call **acldvppGetPicDescRetCode** in the callback function to obtain the return code **retCode**. The value **0** indicates the success of decoding, while **1** indicates failure. If decoding fails, locate the fault based on the return code in the log. For details, see **Return Codes**.

       After the decoding is complete, you are advised to free the memory for storing the input streams and output images of the VDEC, and the corresponding video stream description and image description in the callback function in a timely manner.

       2) The user needs to create a thread in advance and customize a thread function. Calling **aclrtProcessReport** in the thread function triggers the callback function in **1.ii.1)** after a specified period of time.

       **NOTE**

       If **aclvdecSetChannelDescOutPicFormat** is not called to set the output format, images in YUV420 SP NV12 are output by default.

   - The following APIs are encapsulated in **aclvdecCreateChannel** and do not need to be called separately:
     i. **aclrtCreateStream**: explicitly creates a stream. It is internally used for VDEC.
     ii. **aclrtSubscribeReport**: specifies a thread for processing the callback function in a stream. The callback function and thread are specified by calling the **aclvdecSetChannelDesc series**.

2. Decode a video stream into a YUV420SP image by calling **aclvdecSendFrame**.
   - Perform the following steps before decoding a video:
     - Call **acldvppCreateStreamDesc** to create the description of the input video stream, and call **acldvppSetStreamDesc series** to configure the input video, such as the memory address, memory size, and stream format.
     - Call **acldvppCreatePicDesc** to create the description of the output image, and call **acldvppSetPicDesc series** to configure the output image, such as the memory address, memory size, and image format.
- `aclrtLaunchCallback` is encapsulated in the `aclvdecSendFrame` to add a callback function that needs to be executed on the host to the stream task queue. `aclrtLaunchCallback` does not need to be called separately.

3. Call `aclvdecDestroyChannel` to destroy a video processing channel.
   - The channel is destroyed only after the transmitted frames are decoded and the callback function is processed.
   - The following APIs are encapsulated in `aclvdecDestroyChannel` and do not need to be called separately:
     - `aclrtUnSubscribeReport`: unsubscribes a thread. (The callback function in the stream is no longer processed by the specified thread.)
     - `aclrtDestroyStream`: destroys a stream.
   - Call `aclvdecDestroyChannelDesc` to destroy the channel description after the channel is destroyed.
2.4.5 Video Encoding

The user thread tid1 and callback thread tid2 must be created in advance.

Figure 2-8 Video encoding flowchart
The major steps are as follows:

1. Call `aclvencCreateChannel` to create a channel for video encoding.
   - Perform the following steps before creating a channel for video encoding.
     i. Call `aclvencCreateChannelDesc` to create the encoding channel description.
     ii. Call `aclvencSetChannelDesc series` to set the encoding channel description attributes, including the thread, callback function, video encoding protocol, and input image format.

   1) The callback function needs to be created by the user in advance. It is used to obtain the encoding data after video encoding and free resources in a timely manner. For details about the callback function prototype, see 7.12.8.3 `aclvencCallback`.
      - After the encoding is complete, you are advised to free the memory for storing the input images and the corresponding image description in the callback function in a timely manner. The output memory is managed by the system. Therefore, the output memory does not need to be freed by the user.

   2) The user needs to create a thread in advance and customize a thread function. Calling `aclrtProcessReport` in the thread function triggers the callback function in 1.ii.1) after a specified period of time.

   - The following APIs are encapsulated in `aclvencCreateChannel` and do not need to be called separately:
     i. `aclrtCreateStream`: explicitly creates a stream. It is internally used for VENC.
     ii. `aclrtSubscribeReport`: specifies a thread for processing the callback function in a stream. The callback function and thread are specified by calling the `aclvencSetChannelDesc series`.

2. Call `aclvencSendFrame` to encode YUV420SP images into H.264/H.265 video streams.
   - Perform the following steps before encoding a video:
     - Call `acldvppCreatePicDesc` to create the description of the input image, and call `acldvppSetPicDesc series` to configure the input image, such as the memory address, memory size, and image format.
     - Call `aclvencCreateFrameConfig` to create the single-frame configuration data, and call `aclvencSetFrameConfig series` to configure whether to forcibly restart the I-frame interval or end the frame.

   - `aclrtLaunchCallback` is encapsulated in the `aclvencSendFrame` to add a callback function that needs to be executed on the host to the stream task queue. `aclrtLaunchCallback` does not need to be called separately.

3. Call `aclvencDestroyChannel` to destroy a video processing channel.
   - The channel is destroyed only after the transmitted frames are decoded and the callback function is processed.
– The following APIs are encapsulated in `aclvencDestroyChannel` and do not need to be called separately:
  
  ▪ `aclrtUnSubscribeReport`: unsubscribes a thread. (The callback function in the stream is no longer processed by the specified thread.)
  
  ▪ `aclrtDestroyStream`: destroys a stream.

– Call `aclvencDestroyChannelDesc` to destroy the channel description after the channel is destroyed.

2.5 Model Inference
2.5.1 Typical Model Inference Flow

Figure 2-9 Typical model inference flowchart

The major steps are as follows:

1. Call `aclmdlCreateDesc` to create data for describing the model.
2. Call `aclmdlGetDesc` to obtain the model description using the model ID returned in 2.3 Model Loading.
3. Prepare the input and output data for model inference. For details, see 2.5.3 Preparing Input and Output Data for Model Inference.
4. Set dynamic batch/image size and dynamic AIPP. For details, see 2.5.2 Setting Dynamic Batch/Image Size and Dynamic AIPP.
Before setting, ensure that the loaded offline model is configured with
dynamic batch/image size, and dynamic AIPP attributes. For details, see 2.3
Model Loading.

5. Perform model inference.

In fixed-batch scenarios with \texttt{batchSize} > 1, the input data is transferred for
model inference only when the \texttt{batchSize} requirement is met. If not, perform
operations as required.

Currently, synchronous and asynchronous model inference is supported.

- Call \texttt{aclmdlExecute} for synchronous inference.
- Call \texttt{aclmdlExecuteAsync} for asynchronous inference.

For asynchronous decoding, \texttt{aclrtSynchronizeStream} needs to be called
to block the host until all tasks in the specified stream are complete.

Implement the callback function during model inference as required by
referring to 2.7.3 Scenarios with Callback.

6. Obtain the results of model inference for subsequent use.

For synchronous inference, obtain the output data of model inference directly.
For asynchronous inference, when the callback function is implemented,
obtain the model inference result from the callback function for subsequent
use.

7. Free the memory.

Call \texttt{aclrtFree} to free the memory on the device.

8. Free data of specific types.

After the model inference is complete, call \texttt{aclDestroyDataBuffer} and
\texttt{aclmdlDestroyDataset} in sequence to free data types that describe the input
of the model. The \texttt{aclDestroyDataBuffer} API needs to be repeatedly called
for each input or output.

2.5.2 Setting Dynamic Batch/Image Size and Dynamic AIPP

\textbf{NOTICE}

For a single model, dynamic batch size and dynamic image size are mutually
exclusive. That is, \texttt{aclmdlSetDynamicBatchSize} and \texttt{aclmdlSetDynamicHWSize}
cannot be both called.
1. Prepare input data for model inference in dynamic batch/image size, and dynamic AIPP scenarios. For details, see 2.5.3 Preparing Input and Output Data for Model Inference.

Before allocating memory for the dynamic batch/image size and AIPP inputs, call aclmdlGetInputIndexByName to obtain the index of an input using the input name. For dynamic batch or image size scenarios, names are fixed to ACL_DYNAMIC_TENSOR_NAME. For dynamic AIPP scenarios, names are fixed to ACL_DYNAMIC_AIPP_NAME. Call aclmdlGetInputSizeByIndex to obtain the input size using the index of the input.

Data in the memory does not need to be set after memory allocation (otherwise, services may be abnormal). After calling the APIs described in 2, the system automatically sets the data in the memory.

**NOTE**

- Definition of the macro ACL_DYNAMIC_AIPP_NAME is as follows:
  ```c
  #define ACL_DYNAMIC_AIPP_NAME "ascend_dynamic_aipp_data"
  ```

- Definition of the macro ACL_DYNAMIC_TENSOR_NAME is as follows:
  ```c
  #define ACL_DYNAMIC_TENSOR_NAME "ascend_mbatch_shape_data"
  ```

2. Set the dynamic batch/image size choices, and dynamic AIPP attributes after the model is successfully loaded before model execution.

a. Call aclmdlGetInputIndexByName to obtain the index of the input dynamic batch size choices using the input name (names are fixed to ACL_DYNAMIC_TENSOR_NAME).

b. Set dynamic batch/image size choices, and dynamic AIPP attributes.

   - Call aclmdlSetDynamicBatchSize to set the dynamic batch size choices.
- Call `aclmdlSetDynamicHWSize` to set the dynamic image size choices.

- Call the following APIs to set the dynamic AIPP data for model inference.
  1) Call `aclmdlCreateAIPP` to create data of type `aclmdlAIPP`.
  2) Call the APIs provided in 7.15.17.2 Dynamic AIPP Parameters Setting to set dynamic AIPP attributes as required.
     In dynamic AIPP scenarios, `aclmdlSetAIPPSrcImageSize` must be called to set source image size.
  3) Call `aclmdlSetInputAIPP` to set the dynamic AIPP data for model inference.
  4) Call `aclmdlDestroyAIPP` to destroy data of type `aclmdlAIPP` in a timely manner.
2.5.3 Preparing Input and Output Data for Model Inference

Figure 2-11 Input and output data preparation flowchart

Create data to describe the input/output data:
\textit{aclmdlCreateDataset}

Obtain the number of inputs/outputs of a model:
\textit{aclmdlGetNumInputs}
\textit{aclmdlGetNumOutputs}

Obtain the memory size of each input/output:
\textit{aclmdlGetInputSizeByIndex}
\textit{aclmdlGetOutputSizeByIndex}

Allocate memory for each input/output:
\textit{aclrtMalloc}

Create data to describe the input/output memory:
\textit{aclCreateDataBuffer}

Obtain input/output names:
\textit{aclmdlGetInputNameByIndex}
\textit{aclmdlGetOutputNameByIndex}

Add \textit{aclDataBuffer} to \textit{aclmdlDataset}:
\textit{aclmdlAddDatasetBuffer}

Data of type \textit{aclmdlDesc} is used to describe the model, for example, the number of inputs or outputs, names, data types, formats and dimensions. Data of type \textit{aclmdlDataset} is used to describe the input and output data. A model may have multiple inputs and outputs, and data of type \textit{aclDataBuffer} is used to describe the memory address and memory size of each input or output.
Call `aclrtMemcpy` (synchronous mode) or `aclrtMemcpyAsync` (asynchronous mode) to implement data transfer from the host to the device through memory copy.

For model with multiple inputs and outputs, when adding `aclDataBuffer` to the `aclmdlDataset`, you are advised to obtain the names of the inputs and outputs first, and add `aclDataBuffer` in accordance with the corresponding index sequence.

**NOTE**

In fixed-batch scenarios with `batchSize > 1`, before creating data of type `aclDataBuffer`, `aclmdlGetInputSizeByIndex` needs to be called to obtain the required memory size using the index of the input.
2.6 Operator Calling
Figure 2-13 Operator calling flowchart

- Is the operator supported by the system?
  - Y: Develop a custom operator.
  - N: Is it a dynamic-shape operator?
    - Y: Register the dynamic-shape operator selector.
      - aclOpRegisterCompilerFunc
    - N: Generate an .onm file with ATC.
  - N: Load the single-operator model:
    - aclOpSetModelDir
    - aclOpLoad

- Allocate memory to store the operator input/output data: aclrtMalloc

- Whether the operator is encapsulated as an ACL APP
  - Y: Execute the operator using the ACL CBLAS API
  - N: Execute the operator:
    - aclOpExecute
    - aclOpExecWithinHandle

- Perform synchronous wait: aclrtSynchronizeStream

- Free the memory: aclrtFree
The operator calling flow is briefly described as follows (for details about the operators supported by the system, see Operator List):

1. Load and build an operator.
   - For an operator with a fixed shape, call the ACL API to load the operator.
     ▪ Build the single operator definition file (.json) into an offline model supported by the Ascend AI processor (.om file) in advance by referring to Model Conversion Using ATC in ATC Tool Instructions.
     ▪ A single-operator model file can be loaded using the following APIs:
       - `aclopSetModelDir`: sets the directory for loading the model file. The single-operator model file (.om file) is stored in the directory.
       - `aclopLoad`: loads the single-operator model data from the memory. The memory is managed by the user. Single-operator model data refers to the data that is loaded to the memory from the .om file. The .om file is built from a single operator.
   - For an operator with dynamic shape, register the custom operator in advance.
     ▪ Call `aclopRegisterCompileFunc` to register the operator selector (that is, selecting the tiling policy function). Different tiling strategies are adopted for different shapes when executing the operator.
       - The operator selector needs to be defined and implemented by the user in advance. For details about the implementation example of the operator selector, see Reference > TIK Custom Operator with Dynamic Shape in TBE Custom Operator Development Guide.
         ○ Function prototype
           
           ```c
           typedef aclError (*aclopCompileFunc)(int numInputs, const aclTensorDesc *const inputDesc[], int numOutputs, const aclTensorDesc *const outputDesc[], const aclopAttr *opAttr, aclopKernelDesc *aclopKernelDesc);
           ```
         ○ Function implementation
           
           You can write code logic to select a tiling policy and generate tiling parameters, and call `aclopSetKernelArgs` to set tiling arguments and number of blocks for concurrent execution.
     ▪ Call `aclopCreateKernel` to register the operator to the system for code implementation when executing the operator.
     ▪ Call `aclopUpdateParams` to build the operator and trigger the calling logic of the operator selector.

2. Call `aclrtMalloc` to allocate memory on the device to store the input and output data of the operator.
   Call `aclrtMemcpy` (synchronous mode) or `aclrtMemcpyAsync` (asynchronous mode) to implement data transfer from the host to the device through memory copy.

3. Execute the operator.
   - If the operator is GEMM which has been built in the system and encapsulated into an ACL API, you can directly call the CBLAS API to execute the operator.
- If the operator is built in the system but is not encapsulated into an ACL API, the operator can be executed in either of the following ways:
  - Construct the operator description information (such as the input and output tensor description and operator attributes), allocate memory for storing the input and output data of the operator, and call `aclopExecute` to load and execute the operator.
    
    In this mode, the system matches the model in the memory based on the operator description in every `aclopExecute` call to execute the operator.
  
  - Construct the operator description information (such as the input and output tensor description and operator attributes), allocate memory for storing the input and output data of the operator, call `aclopCreateHandle` to create a Handle, and call `aclopExecWithHandle` to load and execute an operator.
    
    In this mode, when `aclopCreateHandle` is called, the system matches the model in the memory based on the operator description, which is cached in the Handle. The Handle improves the efficiency for scenarios where the same operator is executed for multiples times with the `aclopExecWithHandle` call. Call `aclopDestroyHandle` to destroy the handle when it is no longer needed.

- If the operator is not a built-in operator, you need to develop the operator by referring to `TBE Custom Operator Development Guide` and then execute the operator by referring to the description above.

4. Call `aclrtSynchronizeStream` to block application execution until all tasks in the specified stream are complete.

5. Call `aclrtFree` to free the memory.
   
   Call `aclrtMemcp` (synchronous mode) or `aclrtMemcpyAsync` (asynchronous mode) to implement data transfer from the device to the host through memory copy.

### 2.7 Synchronous Wait
2.7.1 Multi-Device Scenario

Figure 2-14 Synchronous wait flowchart in the multi-device scenario

- When there are multiple devices, `aclrtSetCurrentContext` is recommended for switching between the devices, which is more efficient than using `aclrtSetDevice`.
- Call `aclrtSynchronizeDevice` to wait until the computing tasks on the device are complete.
- For details about the model loading flow, see 2.3 Model Loading. For details about the model execution flow, see 2.5 Model Inference.
- For details about the operator loading and execution flow, see 2.6 Operator Calling.

2.7.2 Multi-Stream Scenario

Figure 2-15 Synchronous wait flowchart in multi-stream scenario
Synchronous wait of tasks between streams can be implemented by using events. Call `aclrtStreamWaitEvent` to block the execution of a specified stream until the specified event is complete.

For details about the model loading flow, see 2.3 Model Loading. For details about the model execution flow, see 2.5 Model Inference.

For details about the operator loading and execution flow, see 2.6 Operator Calling.

### 2.7.3 Scenarios with Callback

**Figure 2-16 Synchronous wait flowchart for model inference with callback**

User thread `tid1` and callback thread `tid2` must be created in advance.

- **User thread `tid1`**
  - Create a context: `aclrtCreateContext`
  - Create a stream: `aclrtCreateStream(&stream)`
  - Register the thread for performing callback processing in the stream: `aclrtSubscribeReport(tid2, stream)`
  - Load and execute the model asynchronously: `aclrtExecuteAsync(..., stream)`
  - Add a callback to indicate that the model execution ends: `aclrtLaunchCallback(tid2, aclrtCallback, fn, &userdata, stream)`
  - Execute other code logic

- **Callback thread `tid2`**
  - Loop
  - Whether to trigger callback?
    - Trigger callback function processing in fn to obtain the computation result
    - End callback processing
    - End upon timeout

- Unregister the callback: `aclrtUnsubscribeReport`
- Release runtime resources: `aclrtDestroyEvent`, `aclrtDestroyStream`, `aclrtDestroyContext`
- Deinitialize ACL: `aclrtFinalize`
Figure 2-17 Synchronous wait flowchart for model inference and operator calling with callback

The callback flow is briefly described as follows:

- The user needs to create a callback function in advance to obtain and process the model inference or operator execution result.
- The user needs to create a thread and customize a thread function in advance. Once aclrtProcessReport is called in the thread function, the created callback function will be triggered after a specified period of time.
2.8 Runtime Resource Release

Figure 2-18 Runtime resource release flowchart

- Call aclrtSubscribeReport to specify the thread for processing the callback function in the stream created in advance.
- Call aclrtLaunchCallback to add the created callback function to be executed on the host or device to the task stream.
- Call aclrtUnSubscribeReport to unsubscribe a thread. (The callback function in the stream is no longer processed by the specified thread.)
- In the asynchronous inference scenario with callback processing, to ensure all tasks in a stream are complete, and all inference results are handled by the callback function, aclrtSynchronizeStream needs to be called before destroying the stream.

Release the runtime resources including the streams, contexts and devices in sequence.

- An explicitly created stream is destroyed by using aclrtDestroyStream. An explicitly created context is destroyed by using aclrtDestroyContext. An explicitly specified device using aclrtSetDevice is reset by using aclrtResetDevice.
- If no context or stream is explicitly created, only devices need to be reset by using aclrtResetDevice.
### 2.9 APIs and Data Structures

<table>
<thead>
<tr>
<th>API Type</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL API</td>
<td>See <a href="#">Data Types and Operation APIs</a>.</td>
</tr>
</tbody>
</table>
3 Development Flow

Figure 3-1 Development flowchart

1. Prepare the development environment and operating environment.
2. Analyze development requirements.
   Determine the functions (such as data copy and model inference) required for development based on the specific scenario, and determine the commands and APIs by referring to 5.1 Development Requirement Analysis.

3. Create a code directory.
   Create a directory to store code files, build scripts, test images, and model files by referring to 5.2 Code Directory Creation.

4. Develop an application.
   a. Initialize the ACL, allocate runtime resources, allocate data preprocessing resources, allocate model inference resources, and initialize the single-operator information by referring to 5.4.1 Initializing Resources.
      Before using the ACL APIs to develop an application, you must call the aclInit API to initialize the ACL. Otherwise, errors may occur during the initialization of internal resources, causing other service exceptions.
   b. Copy data from the host to the device by referring to 5.4.2 Copying Data to Device.
   c. Perform data preprocessing (including decoding and resizing) on the device as required by referring to 5.4.3 Pre-Processing Data (Image Decoding and Resizing). Images must be output in YUV420 SP format for model inference.
      After data preprocessing is complete, release related resources.
   d. Perform model inference synchronously by referring to 5.4.4 Performing Model Inference (Batch Size = 1, Fixed Shape, Static AIPP, Single Model).
      After the model inference is complete, release related resources.
   e. Perform data postprocessing as required by referring to 5.4.5 Post-Processing Data Using Single Operator and Returning Result to Host.
      For example, for an image classification application, identify the class labels with the highest confidence scores in data postprocessing.
      After data postprocessing is complete, release related resources.
   f. Release the runtime resources and then deinitialize the ACL by calling the aclFinalize API.

5. Build and run the application, including model conversion, code compilation, and application execution, by referring to 5.5 Application Build and Run.
4 Environment Preparation

For details about how to deploy development environment and operating environment, see the CANN Software Installation Guide of the corresponding Atlas product.

- Once the development environment is deployed, you can obtain the header files required for API calling and the library files required for application building and running.
- After the operating environment is deployed, you can run the executable application files generated after building.
5.1 Development Requirement Analysis

Development Purpose

To develop an image classification application to classify two 1024 x 683 .jpg images.

Implementation Analysis

Analyze the required functions of the application and the involved ACL APIs from the following aspects.
<table>
<thead>
<tr>
<th>Function</th>
<th>Implementation Analysis</th>
<th>Key Command and API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Inference</td>
<td>Select an open-source classification network, for example, the Caffe ResNet-50 network and convert it into an offline model supported by the Ascend AI processor (.om file).</td>
<td>● Ascend Tensor Compiler (ATC) for converting the ResNet-50 into an offline model supported by the Ascend AI processor (.om file); For details, see Model Conversion Using ATC in Model Conversion Using ATC.</td>
</tr>
</tbody>
</table>
|                        |                                                                                                                                                                                                                         | ● API for model loading: aclmdlLoadFromFileWithMem  
|                        |                                                                                                                                                                                                                         | ● API for model execution: aclmdlExecute  
|                        |                                                                                                                                                                                                                         | ● API for model unloading: aclmdlUnload                                                                                                              |
| Data pre-processing    | The ResNet-50 network takes 224 x 224 RGB images only. Therefore, the existing 1024 x 683 .jpg images need to be decoded into the YUV420SP format, resized to 224 x 224, and then converted into the RGB format.                  | ● APIs for decoding and resizing:  
| (including decoding,   |                                                                                                                                                                                                                         |   - acldvppCreateChannelAsync  
| cropping, resizing,    |                                                                                                                                                                                                                         |   - acldvppJpegDecodeAsync  
| encoding, CSC)         |                                                                                                                                                                                                                         |   - acldvppVpcResizeAsync  
|                        |                                                                                                                                                                                                                         |   - acldvppDestroyChannelAsync  
<p>|                        |                                                                                                                                                                                                                         | ● CSC: Specify the --insert_op_conf parameter when using the ATC for model conversion and save the CSC settings in the .om file for calling during model execution. For details, see AIPP Configuration in AIPP Configuration. |
|                        |                                                                                                                                                                                                                         |                                                                                                                                                                                                                                                                                                                                                                          |</p>
<table>
<thead>
<tr>
<th>Function</th>
<th>Implementation Analysis</th>
<th>Key Command and API</th>
</tr>
</thead>
</table>
| Data post-processing| Post-process the model inference result. That is, call the Cast and ArgMaxD operators to identify the class label with the highest confidence score of each image from the inference result. Call the Cast operator to convert the data type of the inference result from float32 to float16, and then call the ArgMaxD operator to identify the class label with the highest confidence score of each image from the inference result. | • Build the .json definition files of the Cast and ArgMaxD operators into an offline model supported by the Ascend AI processor (.om file) respectively to verify the single operators. For details, see Model Conversion Using ATC in Model Conversion Using ATC.  
• Operator execution: aclopExecute |
| Data copy           | Copy image from the host to the device for processing and copy the processing result to the host after processing is complete.                                                                                                                                                                                                                     | API for data copy: aclrtMemcpy                                                                               |

### 5.2 Code Directory Creation

Create a directory to store code files, build script, test images, and model files.

The following is an example of the directory structure:

- App name
  - caffe_model  // directory of the configuration file and model file for model conversion.
    - xxx.cfg
    - xxx.prototxt
  - data
    - xxx.jpg  // test data
  - inc  // directory of header files that declare functions.
    - xxx.h
  - out
    - op_models  // directory of the operator .om file
    - App executable file generated after build
    - src  // directory of the build script and function implementation files
      - xxx.json  // configuration file for system initialization
      - CMakeLists.txt  // build script
      - xxx.cpp
5.3 Model Conversion Using ATC

The Ascend 310 chip is capable of accelerating inference under the Caffe and TensorFlow framework models. During model conversion, operator scheduling tuning, weight data rearrangement, quantization compression, and memory usage tuning can be implemented, and model preprocessing can be completed without using devices. After model training is complete, you need to convert the trained model to the model file (.om file) supported by Ascend 310 by using ATC, compile service code, and call APIs provided by ACL to implement service functions.

Preparations

1. Obtain the ATC software package.
   Before using the ATC, install the development kit Ascend-Toolkit
2. Set environment variables.
   a. Mandatory environment variables: ({$install path} in the following environment variables uses the default installation path /usr/local/Ascend/ascend-toolkit/latest/xxx-linux_gcc.x.x of the development kit Ascend-Toolkit as an example.)
      export PATH=/usr/local/python3.7.5/bin:${install_path}/atc/ccce_compiler/bin:${install_path}/atc/bin:$PATH
      export PYTHONPATH=${install_path}/atc/python/site-packages/te:${install_path}/atc/python/site-packages/topi:$PYTHONPATH
      export LD_LIBRARY_PATH=${install_path}/atc/lib64:$LD_LIBRARY_PATH
      export ASCEND_OPP_PATH=${install_path}/opp
   b. Set optional environment variables:
      i. To use the auto tuning function --auto_tune_mode="RL,GA", modify the environment variables PYTHONPATH and
         LD_LIBRARY_PATH:
         export PYTHONPATH=${install_path}/atc/python/site-packages/te:${install_path}/atc/python/site-packages/auto_tune.egg/auto_tune/:$PYTHONPATH
         export LD_LIBRARY_PATH=${install_path}/acllib/lib64:${install_path}/atc/lib64:$LD_LIBRARY_PATH
      ii. If the slog process exists in the current environment (which can be checked by running the ps -ef | grep slog command), logs are generated and recorded in log files by default. To display logs on the screen or redirect logs to files, set the following environment variable:
         export SLOG_PRINT_TO_STDOUT=1
         If the slog process does not exist in the current environment, logs are displayed on the screen by default. To redirect logs to a file, set the environment variable SLOG_PRINT_TO_STDOUT to 1.
      iii. If you want to convert only TBE operators using single-operator .json files, set the following environment variable:
         export ASCEND_ENGINE_PATH=${install_path}/atc/lib64/plugin/opskernel/libfe.so:${install_path}/atc/lib64/plugin/opskernel/libge_local_engine.so
         After the preceding command is executed, if you need to run the command for converting a third-party model to an offline model again, as shown in Example (Converting a Caffe Model into an Offline Model), run the unset ASCEND_ENGINE_PATH command to
make the **ASCEND_ENGINE_PATH** environment variable take effect first.

iv. If the model is large, you can set the following environment variable to enable parallel operator building during model conversion:

```bash
export TE_PARALLEL_COMPILER=xx
```

The value of **TE_PARALLEL_COMPILER** indicates the number of parallel operator building processes. The value must be an integer. The default value is **8**. As long as the value is greater than **0**, parallel building is enabled. In the single-chip scenario, you are advised to set this parameter to \((80\% \times \text{Host CPU core count})\). In the multi-chip scenario, you are advised to set this parameter to \((80\% \times \text{Host CPU core count/Chip count})\).

**NOTE**

- Environment variables set by using `export` commands are valid only in the current window. If the environment variable of the ATC installation path has been set in the `.bashrc` file, you need to manually delete it before running the preceding commands.
- If it takes too long to convert a model in the Arm (AArch64) development environment, fix it by referring to [9.1 What Do I Do If Model Conversion Takes Too Long When the OS and Architecture Configuration of the Development Environment Is Arm (AArch64)]().

**Example (Converting a Caffe Model into an Offline Model)**

**Step 1**  
Log in to the development environment as the ATC running user and upload the model file (*.prototxt) and weight file (*.caffemodel) to be used for model conversion to any path in the development environment, for example, `$HOME/test/`.

**Step 2**  
Generate a model file (the directory and file arguments in the command are for reference only):

```bash
atc --model=$HOME/test/resnet50.prototxt --weight=$HOME/test/resnet50.caffemodel --framework=0 --output=$HOME/test/out/caffe_resnet50 --soc_version=Ascend310
```

**Step 3**  
If the following message is displayed, the model is successfully converted:

```
ATC run success
```

After the command is executed successfully, you can view the model file (for example, `caffe_resnet50.om`) in the path specified by the `output` parameter.

**NOTE**

If the user model contains custom operators, develop and deploy custom operators by referring to the *TBE Custom Operator Development Guide*. During model conversion, the custom operator library will be preferentially looked up than the built-in operator library for operators in the user model.

**Step 4**  
(Optional) If the output node is specified by using the `--out_nodes` parameter during model conversion, the output information of the operator will not be available after the model is converted into an .om model. You can run the following command to convert the .om model into a .json file and view the output information:

```bash
atc --mode=1 --om=$HOME/test/caffe_resnet50.om --json=$HOME/test/out/resnet.json
```

In the example shown in **Figure 5-1**, the `--out_nodes` parameter specifies the res4f operator as the output node. The left part shows the code without specifying
the `--out_nodes` parameter, and the right part shows the code with the `--out_nodes` parameter specified.

**Figure 5-1** Code with and without the `--out_nodes` parameter specified

```json
"input_i": [3287552],
"input_name": [
  "prob"
],
"name": "Node_Output",
"output_desc": [
  
],
"output_i": [4993696],
"src_index": [0],
"src_name": [
  "prob"
],
"type": "NetOutput"
```

```
"input_i": [1681408,
  75264],
"input_name": [
  "res1f",
  "prob"
],
"name": "Node_Output",
"output_desc": [
  
],
"output_i": [5094912,
  5898240],
"src_index": [0,
  0],
"src_name": [
  "res1f",
  "prob"
],
"type": "NetOutput"
```

----End

**Example (Converting a Single-Operator JSON File into an Offline Model)**

**Step 1** Log in to the development environment as the ATC running user and upload the single-operator JSON file to the `$HOME/singleop` directory in the development environment. This section uses the single-operator GEMM with format ND as an example.

**Step 2** Generate a model file (the directory and file arguments in the command are for reference only):

```
atc --singleop=$HOME/singleop/gemm.json --output=$HOME/test/out/op_model --soc_version=Ascend310
```

**Step 3** If the following message is displayed, the model is successfully converted:

```
ATC run success
```

After the command is executed successfully, you can view the model file, for example, `0_GEMM_1_2_16_1_2_16_1_2_16_1_2_16_1_2_12_12_16_16.om`, in the path specified by the `output` parameter.
The naming rule of the generated offline model file is 
**SN_opType_InputDescription (dataType_format_shape)_OutputDescription (dataType_format_shape)**. The enumeration of **dataType** in the .json file is as follows:

```c
typedef enum {
    ACL_DT_UNDEFINED = -1, // Unknown data type (default)
    ACL_FLOAT = 0,
    ACL_FLOAT16 = 1,
    ACL_INT8 = 2,
    ACL_INT32 = 3,
    ACL_UINT8 = 4,
    ACL_INT16 = 6,
    ACL_UINT16 = 7,
    ACL_UINT32 = 8,
    ACL_INT64 = 9,
    ACL_UINT64 = 10,
    ACL_DOUBLE = 11,
    ACL_BOOL = 12,
} aclDataType;
```

---End

**Example (Converting a TensorFlow Model into an Offline Model)**

**Step 1** Log in to the development environment as the ATC running user and upload the .pb model file to any path in the development environment, for example, `$HOME/test/`.

**Step 2** Generate a model file (the directory and file arguments in the command are for reference only):

- If the original model has a fixed shape, that is, the values of all NHWC parameters in `input_name` are fixed, the conversion command is as follows:
  ```bash
  atc --model=$HOME/test/resnet18_tensorflow.pb --framework=3 --output=$HOME/test/out/tf_resnet18 --soc_version=Ascend310
  ```

- If the original model has a dynamic shape, for example, `input_name1:? ,h,w,c`. In this scenario, `--input_shape` is required, and set `?` to a fixed value as required. The conversion command is as follows:
  ```bash
  atc --model=$HOME/test/module_tensorflow.pb --input_shape="input_name1:n,h,w,c" --framework=3 --output=$HOME/test/out/module_tf --soc_version=Ascend310
  ```

**Step 3** If the following message is displayed, the model is successfully converted:

ATC run success

After the command is executed successfully, you can view the model file (for example, `tf_resnet18.om`) in the path specified by the `output` parameter.

----End

**Example (Converting a MindSpore Model to an Offline Model)**

**Step 1** Log in to the development environment as the ATC running user and upload the .pb model file to any path in the development environment, for example, `$HOME/test/`.

**Step 2** Generate a model file (the directory and file arguments in the command are for reference only):

```bash
atc --model=$HOME/test/ReLU.pb --framework=1 --output=$HOME/test/out/ReLU_mindspore --soc_version=Ascend310
```
Step 3 If the following message is displayed, the model is successfully converted:

ATC run success

After the command is executed successfully, you can view the model file (for example, `ReLU_mindspore.om`) in the path specified by the `output` parameter.

---End

5.4 Application Development

5.4.1 Initializing Resources

5.4.1.1 Initializing ACL

Basic Principles

Call the `aclInit` API to initialize the ACL. The configuration file is in JSON format. You can also configure dump data as described in the Configuration File Example part. For details about the configuration, see Preparing Data > Preparing Dump Data for an Offline Model in Model Accuracy Analyzer Instructions (CLI).

If no configuration is involved, pass a null pointer to the `aclInit` API. The following is an example.

```c
aclError ret = aclInit(nullptr);
```

Sample Code

After the API is called, add an exception handling branch and specify log printing of different levels (such as `ERROR_LOG` and `INFO_LOG`).

A sample code snippet is provided as follows. You can view the complete sample code in the `src/sample_process.cpp` file of the `acl_dvpp_resnet50` sample.

```c
#include "acl/acl.h"
#include "../src/acl.json"

//......
//Initialize basic configurations.
//... indicates a relative path, which is relative to the directory of the executable file.
//For example, if the executable file is stored in the out directory, .. indicates the upper-level directory of the out directory.
const char *aclConfigPath = "../src/acl.json";

aclError ret = aclInit(aclConfigPath);
//......
```

Configuration File Example

Take the Caffe ResNet-50 network as an example. To compare the output results of certain operator layers between Caffe ResNet-50 and the offline model adapted to the Ascend AI Processor converted from Caffe ResNet-50, add the following content in the `acl.json` configuration file:

```json
{
    "dump":{
        "dump_list":{
            "model_name":"ResNet-50",
```
5.4.1.2 Allocating Runtime Resources (Single Process, Single Thread, Single Stream)

The multi-thread and multi-stream scenario is described in 6.1 Stream Management.

Basic Principles

Allocate devices, contexts and streams in sequence to support the execution of computing and management tasks.

- Context and stream:
  - A context explicitly created using aclrtCreateContext is destroyed using aclrtDestroyContext.
  - A context explicitly created using aclrtCreateStream is destroyed using aclrtDestroyStream.
  - A default context or stream created implicitly with the call to aclrtSetDevice has the following restrictions:
    - One device corresponds to one default context. The default context does not need to be destroyed by calling aclrtDestroyContext.
    - One device corresponds to one default stream. The default stream does not need to be destroyed by calling aclrtDestroyStream. To pass the default stream to any API call, pass NULL directly.

- Single-process, single-thread, single-stream scenario:
  - Single-process: One application maps to one process.
  - Single-thread: Only one thread is available by default unless you create more explicitly.
  - Single-stream: A single stream is used in the entire development process. For asynchronous tasks in the same stream, the ACL executes the tasks in its original order.

Sample Code

After the API is called, add an exception handling branch and specify log printing of different levels (such as ERROR_LOG and INFO_LOG).

A sample code snippet is provided as follows. You can view the complete sample code in the src/sample_process.cpp file of the acl_dvpp_resnet50 sample.

```c++
#include "acl/acl.h"

//......
```
//1. Specify the device for the computation.
ret = aclrtSetDevice(deviceId_);

//2. Explicitly create a context to manage the stream objects.
ret = aclrtCreateContext(&context_, deviceId_);

//3. Explicitly create a stream.
// To reserve the execution order of asynchronous tasks.
ret = aclrtCreateStream(&stream_);

//......

5.4.1.3 Allocating Data Pre-processing Resources

Basic Principles

Before data pre-processing (such as image decoding and resizing), you need to create a channel for processing image data and set the resizing algorithm to nearest neighbor interpolation.

Sample Code

After the API is called, add an exception handling branch and specify log printing of different levels (such as ERROR_LOG and INFO_LOG). A sample code snippet is provided as follows.

You can view the complete sample code in the src/sample_process.cpp and src/dvpp_process.cpp files of the acl_dvpp_resnet50 sample.

```c++
#include "acl/acl.h"
//......
//1. Create description of the data processing channel. dvppChannelDesc_ is of type aclDvppChannelDesc.
dvppChannelDesc_ = aclDvppCreateChannelDesc();

//2. Create an data processing channel in asynchronous mode.
aclError ret = aclDvppCreateChannel(dvppChannelDesc_);

//3. Specify the resizing algorithm: resizeConfig_ is of type aclDvppResizeConfig.
resizeConfig_ = aclDvppCreateResizeConfig(0);
//......
```

5.4.1.4 Allocating Model Inference Resources

Basic Principles

Before model inference, you need to load model data from the offline model adapted to the Ascend AI Processor to the memory.

- Model conversion:
  Before model loading, you need to convert the third-party network (for example, Caffe ResNet-50) into an offline model adapted to the Ascend AI Processor by referring to Model Conversion Using ATC in ATC Tool Instructions.

  If CSC is required, you also need to configure the configuration file by referring to AIPP Configuration in ATC Tool Instructions.

- Memory size:
If the memory is managed by the user, you need to call the `aclmdlQuerySize` API to query the sizes of the working memory and weight memory required for model execution to avoid memory waste.

- **Model loading:**
  Model data can be loaded using the following APIs:
  - `aclmdlLoadFromFile`: loads offline model data from a file. The memory is managed by the system.
  - `aclmdlLoadFromMem`: loads offline model data from the memory. The memory is managed by the system.
  - `aclmdlLoadFromFileWithMem`: loads offline model data from a file. The memory (including working memory and weight memory) is managed by the user.
  - `aclmdlLoadFromMemWithMem`: loads offline model data from the memory. The memory (including working memory and weight memory) is managed by the user.

- **Model input and data**
  `aclmdlDataset` is used to describe input and output data during model inference. A model may have multiple inputs and outputs, and data of the `aclDataBuffer` type is used to describe the memory address and memory size of each input or output.

### Sample Code

After the API is called, add an exception handling branch and specify log printing of different levels (such as `ERROR_LOG` and `INFO_LOG`). A sample code snippet is provided as follows.

A sample code snippet is provided as follows. You can view the complete sample code in the `src/sample_process.cpp` and `src/model_process.cpp` files of the `acl_dvpp_resnet50` sample.

```cpp
#include "acl/acl.h"
//......
//1. Initialize variables.
//... indicates a relative path, which is relative to the directory of the executable file.
//For example, if the executable file is stored in the out directory, .. indicates the upper-level directory of the out directory.
const char* omModelPath = "../model/resnet50_aipp.om";
//......

//2. Obtain the weight memory size and working memory size required for model execution.
aclError ret = aclmdlQuerySize(omModelPath, &modelMemSize_, &modelWeightSize_);

//3. Allocate the working memory for executing the model on the device based on the working memory size obtained in step 2.
ret = aclrtMalloc(&modelMemPtr_, modelMemSize_, ACL_MEM_MALLOC_NORMAL_ONLY);

//4. Allocate the weight memory for executing the model on the device based on the working memory size obtained in step 2.
ret = aclrtMalloc(&modelWeightPtr_, modelWeightSize_, ACL_MEM_MALLOC_NORMAL_ONLY);

//5. Load the offline model adapted to the Ascend AI Processor. The memory (including the weight memory and working memory) is managed by the user.
// The model is successfully loaded, and the model ID is returned.
ret = aclmdlLoadFromFileWithMem(modelPath, &modelId_, modelMemPtr_, modelMemSize_, modelWeightPtr_, modelWeightSize_);

//6. Obtain the model description based on the model ID returned in step 5.
```
//modelDesc_ is of type aclmdlDesc.
modelDesc_ = aclmdlCreateDesc();
ret = aclmdlGetDesc(modelDesc_, modelId_);

//7. Create data of type aclmdlDataset to describe the output of model inference.
//output_ is of type aclmdlDataset.
output_ = aclmdlCreateDataset();

//7.1 Obtain the number of outputs.
size_t outputSize = aclmdlGetNumOutputs(modelDesc_);

//7.2 Allocate memory for each output with the for loop and add each output to the data of type aclmdlDataset.
for (size_t i = 0; i < outputSize; ++i) {
    size_t buffer_size = aclmdlGetOutputSizeByIndex(modelDesc_, i);
    void *outputBuffer = nullptr;
    aclError ret = aclrtMalloc(&outputBuffer, buffer_size, ACL_MEM_MALLOC_NORMAL_ONLY);
    aclDataBuffer* outputData = aclCreateDataBuffer(outputBuffer, buffer_size);
    ret = aclmdlAddDatasetBuffer(output_, outputData);
}

5.4.1.5 Initializing Single-Operator Information

Basic Principles

In the data post-processing phase, the Cast operator is called to convert the data type of the inference result from float32 to float16, and the ArgMaxD operator is called to identify the class label with the highest confidence score of each image from the inference result. Therefore, you need to build the .json definition files of the Cast and ArgMaxD operators into an offline model supported by the Ascend AI processor (.om file) respectively to verify the single operators in advance. For details, see ATC Tool Instructions.

The path for storing the converted .om files must be the same as that passed to the aclopSetModelDir call.

Sample Code

After the API is called, add an exception handling branch and specify log printing of different levels (such as ERROR_LOG and INFO_LOG).

A sample code snippet is provided as follows. You can view the complete sample code in the src/sample_process.cpp and src/singleOp_process.cpp files of the acl_dvpp_resnet50 sample.

```c
#include "acl/acl.h"

// Set the directory of the single-operator model file.
/acldir relative to the directory of the executable file. For example, if the executable file is stored in the out directory, the directory is out/op_models.
acError ret = aclopSetModelDir("op_models");
```

5.4.2 Copying Data to Device

Basic Principles

Load image data to the memory, and then copy image data from the host memory to the device memory using aclrtMemcpy for processing. The aclrtMemcpyAsync API implements asynchronous data copy.
● If data pre-processing (such as JPG image decoding and resizing) is required on the device, the data pre-processing memory must be allocated using `acldvppMalloc`.

● If data pre-processing is not required on the device, the device memory is allocated using `aclrtMalloc`.

### Sample Code

After the API is called, add an exception handling branch and specify log printing of different levels (such as `ERROR_LOG` and `INFO_LOG`).

A sample code snippet is provided as follows. You can view the complete sample code in the `src/sample_process.cpp`, `src/dvpp_process.cpp`, and `src/utils.cpp` files of the `acl_dvpp_resnet50` sample.

```cpp
#include <iostream>
#include <fstream>
#include <cstring>
#include "acl/acl.h"
#include "acl/ops/acldvpp.h"

// PicDesc is a user-defined structure that describes the image path and image name (picName), and image width and height.
PicDesc testPic[] = {
    { "../data/dog1_1024_683.jpg", 1024, 683 },
    { "../data/dog2_1024_683.jpg", 1024, 683 },
};

// Process each image with the for loop.
for (size_t index = 0; index < sizeof(testPic) / sizeof(testPic[0]); ++index) {
    // 1. Defined the function. Call the function in the C++ standard library `std::ifstream` to read the image file and output the buffer size `inputHostBuffSize` occupied by the image file. The buffer address `inputHostBuff` of the image file on the host is returned.
    char* inputHostBuff = ReadBinFile(testPic.picName,
    inputHostBuffSize);

    // 2. Allocate the decoding input buffer.
    void* inBufferDev = nullptr;
    uint32_t inBufferSize = inputHostBuffSize;
    aclError ret = acldvppMalloc(&inBufferDev, inBufferSize);

    // 3. Copy the data memory from the host to the device.
    ret = aclrtMemcpy(inBufferDev, inBufferSize, inputHostBuff,
    ACL_MEMCPY_HOST_TO_DEVICE);
    // To-do: data pre-processing (image decoding and resizing)
}
```

5.4.3 Pre-Processing Data (Image Decoding and Resizing)

### Basic Principles

After data is copied to the device, the device decodes and resizes the image and outputs the image in source format or YUV420 SP format as the input for model inference.

- Decoding:
  - The JPEGD decodes and outputs images in the following formats:
    - jpeg (YUV444SP) -> YUV444SP with V component before U component, YUV420SP with V component before U component, or YUV420SP with U component before V component
- jpeg (YUV422SP) -> YUV422SP with V component before U component, YUV420SP with V component before U component, or YUV420SP with U component before V component
- jpeg (YUV420SP) -> YUV420SP with V component before U component, or YUV420SP with U component before V component
- jpeg (YUV400) -> YUV420SP with U and V components padded with 0 x 80
  - The width and height of the decoded image must be rounded up to 128 pixels and 16 pixels, respectively.
  - The memory for storing the decoded image is calculated as follows:
    - YUV420SP: Rounded-up width x Rounded-up height x 3/2
    - YUV422SP: Rounded-up width x Rounded-up height x 2
    - YUV444SP: Rounded-up width x Rounded-up height x 3

- Resizing:
  - The width and height of the image to be resized must be rounded up to 16 pixels and 2 pixels, respectively.
  - The memory for storing the resized image is calculated based on the YUV420SP format as follows: Rounded-up width x Rounded-up height x 3/2

- Resource Release:
  - After data pre-processing is complete, resources such as the input memory of data pre-processing and image description information, except the output memory (which is used as the input of model inference and can be released only after model inference is complete), need to be released in a timely manner.
  - The memory related to data pre-processing must be allocated using the acldevppMalloc API and freed through the acldevppFree API.

For details about other data pre-processing functions, see 6.4 Data Preprocessing.

Sample Code

After the API is called, add an exception handling branch and specify log printing of different levels (such as ERROR_LOG and INFO_LOG).

A sample code snippet is provided as follows. You can view the complete sample code in the src/sample_process.cpp and src/dvpp_process.cpp files of the acl_dvpp_resnet50 sample.

```c
#include "acl/acl.h"
#include "acl/ops/acl_dvpp.h"

//......
// Initialize variables. For details, see the sample code.

//Process each image with the for loop. testPic is a user-defined structure that describes the image name picName, and image width and height.
for (size_t index = 0; index < sizeof(testPic) / sizeof(testPic[0]); ++index) {

    // 1. Allocate the resizing output buffer resizeOutBufferDev.
    // The memory for storing the resized image is calculated based on the YUV420SP format as follows: Rounded-up width x
```
// 2. Allocate the output buffer for storing decoded image and create the description of the decoded image.

// 2.1 Allocate the decoding output buffer decodeOutDevBuffer_, decodeOutBufferSize is calculated based on the YUV420SP format as follows: Rounded-up width x Rounded-up height x 3/2
ret = acldvppMalloc(&decodeOutDevBuffer_, decodeOutBufferSize);

// 2.2 Create the description of the decoded image and set the attribute values.
decodeOutputDesc_ = acldvppCreatePicDesc();
acldvppSetPicDescData(decodeOutputDesc_, decodeOutDevBuffer_);
acldvppSetPicDescFormat(decodeOutputDesc_, PIXEL_FORMAT_YUV_SEMIPLANAR_420);
acldvppSetPicDescWidth(decodeOutputDesc_, inputWidth_);
acldvppSetPicDescHeight(decodeOutputDesc_, inputHeight_);
acldvppSetPicDescWidthStride(decodeOutputDesc_, inputStride_);
acldvppSetPicDescHeightStride(decodeOutputDesc_, inputStride_);
acldvppSetPicDescSize(decodeOutputDesc_, decodeOutBufferSize);

// 3. Perform asynchronous decoding and call aclrtSynchronizeStream to block the host until all tasks in the specified stream are complete.
ret = aclrvppJpegDecodeAsync(dvppChannelDesc_, inDevBuffer_, inDevBufferSize_, decodeOutputDesc_, stream_);
ret = aclrtSynchronizeStream(stream_);

// 4. After decoding is complete, release resources, including the description of the decoded output image.
acldvppDestroyPicDesc(decodeOutputDesc_);

// 5. Create the description of the image to be resized and set the attribute values.
resizeInputDesc_ = acldvppCreatePicDesc();
acldvppSetPicDescData(resizeInputDesc_, decodeOutDevBuffer_);
acldvppSetPicDescFormat(resizeInputDesc_, PIXEL_FORMAT_YUV_SEMIPLANAR_420);
acldvppSetPicDescWidth(resizeInputDesc_, inputWidth_);
acldvppSetPicDescHeight(resizeInputDesc_, inputHeight_);
acldvppSetPicDescWidthStride(resizeInputDesc_, jpegOutWidthStride);
acldvppSetPicDescHeightStride(resizeInputDesc_, jpegOutHeightStride);
acldvppSetPicDescSize(resizeInputDesc_, jpegOutBufferSize);

// 6. Create the description of the resized image. The data in the resizeOutBufferDev_ buffer is used as the input of model inference.
resizeOutputDesc_ = acldvppCreatePicDesc();
acldvppSetPicDescData(resizeOutputDesc_, resizeOutBufferDev_);
acldvppSetPicDescFormat(resizeOutputDesc_, PIXEL_FORMAT_YUV_SEMIPLANAR_420);
acldvppSetPicDescWidth(resizeOutputDesc_, modelInputWidth_);
acldvppSetPicDescHeight(resizeOutputDesc_, modelInputHeight_);
acldvppSetPicDescWidthStride(resizeOutputDesc_, resizeOutputWidthStride);
acldvppSetPicDescHeightStride(resizeOutputDesc_, resizeOutputHeightStride);
acldvppSetPicDescSize(resizeOutputDesc_, resizeOutBufferSize_);

// 7. Perform asynchronous resizing and call aclrtSynchronizeStream to block the host until all tasks in the specified stream are complete.
ret = aclrvppVpcResizeAsync(dvppChannelDesc_, resizeInputDesc_, resizeOutConfig_, stream_);
ret = aclrtSynchronizeStream(stream_);

// 8. Release resources.
// 8.1 After the resizing is complete, release resources, including the decoding output buffer, description of the image to be resized, and description of the resized image.
(void) acldvppFree(decodeOutDevBuffer_);
acldvppDestroyPicDesc(resizeInputDesc_);
acldvppDestroyPicDesc(resizeOutputDesc_);
// 8.2 Release other resources, including the resizing configuration, data processing channel, and description.
acldvppDestroyResizeConfig(resizeConfig_);
aclError ret = aclrvppDestroyChannel(dvppChannelDesc_);
5.4.4 Performing Model Inference (Batch Size = 1, Fixed Shape, Static AIPP, Single Model)

**Basic Principles**

Perform model inference synchronously. After the model inference is complete, release related resources.

- **Model inference:**
  
  Pass the model ID along with other required arguments to the `aclmdlExecute` call to implement synchronous model inference. The model ID is obtained from the return values of resource allocation for model inference.

- **Resource release:**
  
  - The data pre-processing output is used as the input of model inference. After model inference is complete, call `acldvppFree` to free the memory in a timely manner.
  
  - `aclmdlDataset` is used to describe input and output data during model inference. A model may have multiple inputs and outputs, and data of the `aclDataBuffer` type is used to describe the memory address and memory size of each input or output. After the model inference is complete, call `aclDestroyDataBuffer` and `aclmdlDestroyDataset` in sequence to release the input data of the model. The `aclDestroyDataBuffer` API needs to be repeatedly called for each input or output.
  
  - After the model inference is complete, call `aclmdlUnload` to unload the model.

**Sample Code**

After the API is called, add an exception handling branch and specify log printing of different levels (such as `ERROR_LOG` and `INFO_LOG`).

A sample code snippet is provided as follows. You can view the complete sample code in the `src/sample_process.cpp` and `src/model_process.cpp` files of the `acl_dvpp_resnet50` sample.

```c
#include "acl/acl.h"

//Process each image with the for loop. testPic is a user-defined structure that describes the image name picName, and image width and height.
for (size_t index = 0; index < sizeof(testPic) / sizeof(testPic[0]); ++index) {
    // 1. Create input data for model inference.
    aclmdlDataset *input_ = aclmdlCreateDataset();
    aclError ret = aclmdlAddDatasetBuffer(input_, inputData);
    //The model in the sample has only one input. The resizing output buffer is used as the input of model inference. inputDataBuffer indicates the resizing output buffer, and bufferSize indicates the buffer size. aclDataBuffer* inputData = aclCreateDataBuffer(inputDataBuffer, bufferSize);
    aclDestroyDataBuffer(inputDataBuffer);
}```
//  
modelId_ indicates the model ID. After the model is successfully loaded during resource initialization of model inference, the model ID is returned.
//output_ indicates the output data of model inference, which has been defined during resource initialization of model inference.

ret = aclmdlExecute(modelId_, input_, output_)

// 3. After model inference is complete, release the pre-processing output buffer dvppOutputBuffer.
acldvppFree(dvppOutputBuffer);

// 4. Release resources.
// If the model has multiple inputs, repeatedly call aclDestroyDataBuffer for each input to destroy the data of type aclmdlDataset
aclDestroyDataBuffer(inputData);
acldmlDestroyDataset(input_);
//To-do: data post-processing
).

5.4.5 Post-Processing Data Using Single Operator and Returning Result to Host

Data Post-processing (Image Classification)

After model inference is complete, post-process the model inference result. After data post-processing is complete, release related resources.

In the current sample, the Cast operator is used to convert the data type of the inference result from float32 to float16, and the ArgMaxD operator is used to identify the class label with the highest confidence score of each image from the inference result.

- The Cast operator has been encapsulated into an ACL API. You can directly pass the input and output tensor description and the memory of the input and output data of the operator to the aclopCast call to load and execute the operator.
- The ArgMaxD operator has not been encapsulated into an ACL API. Therefore, you must construct the operator description information (including the input and output tensor description and operator attributes), allocate the memory for storing the input and output data of the operator, specify the operator type, and call the aclopExecute API to load and execute the operator.

For details about the usage of other single operators, see 6.7 Single-Operator Calling.

Sample Code

After the API is called, add an exception handling branch and specify log printing of different levels (such as ERROR_LOG and INFO_LOG).

A sample code snippet is provided as follows. You can view the complete sample code in the src/sample_process.cpp and src/singleOp_process.cpp files of the acl_dvpp_resnet50 sample.

#include "acl/acl.h"

//......

// Process each image with the for loop. testPic is a user-defined structure that describes the image name picName, and image width and height.
for (size_t index = 0; index < sizeof(testPic) / sizeof(testPic[0]); ++index) {
  // 1. Before data post-processing, obtain the output of model inference: modelOutput.
  aclDataBuffer inputBuffer_[0] = aclmdlGetDatasetBuffer(modelOutput, 0);

  // 2. Define the RunSingleOpCast function to construct the input and output tensor description of the Cast
  // operator, allocate the buffer devBufferCast_ for storing the operator output data, and call the aclopCast
  // API to load and execute the operator.
  Result ret = RunSingleOpCast();

  // 3. Define the RunSingleOpArgMaxD function, construct the input and output tensors, input and output
  // tensor description, and operator attributes of the ArgMaxD operator, allocate the buffer
  // devBufferArgMaxD_ for storing the operator output data, and call the aclopExecute API to load and
  // execute the operator.
  ret = RunSingleOpArgMaxD();

  // 4. Return the ArgMaxD output to the host.
  // 4.1 Allocate memory on the host based on the size of the ArgMaxD output.
  void* hostBuffer = nullptr;
  aclError ret = aclrtMallocHost(&hostBuffer, tensorSizeArgMaxD_);

  // 4.2 Copy the ArgMaxD output from the device to the host.
  ret = aclrtMemcpy(hostBuffer, tensorSizeArgMaxD_, devBufferArgMaxD_,
                    tensorSizeArgMaxD_, ACL_MEMCPY_DEVICE_TO_HOST);

  // 4.3 Print the class label with the highest confidence score in the terminal console.
  int32_t* index = static_cast<int32_t*>(hostBuffer);
  INFO_LOG("--> index of classification result is %d", *index);

  // 5. Release resources.
  // 5.1 Free the memory on the host.
  aclrtFreeHost(hostBuffer);

  // 5.2 Free the memory for storing the operator output on the device.
  (void)aclrtFree(devBufferCast_);
  (void)aclrtFree(devBufferArgMaxD_);

  // 5.3 Destroy the data of type aclDataBuffer (used to describe the operator output).
  (void)aclDestroyDataBuffer(outputBufferCast_[0]);
  (void)aclDestroyDataBuffer(outputBufferArgMaxD_[0]);

  // ... } //To-do: Release runtime resources.

5.4.6 Releasing Runtime Resources and Deinitializing ACL

Basic Principles

After all data on the host and device is processed, release the runtime resources, including the streams, contexts and devices in sequence.

- The default context cannot be released using aclrtDestroyContext.
- The default stream cannot be released using aclrtDestroyStream.
- The default context and stream are automatically released with the reset of device using aclrtResetDevice.
- Deinitialize ACL by calling aclFinalize.

Sample Code

After the API is called, add an exception handling branch and specify log printing of different levels (such as ERROR_LOG and INFO_LOG).

A sample code snippet is provided as follows. You can view the complete sample code in the src/sample_process.cpp file of the acl_dvpp_resnet50 sample.
#include "acl/acl.h"

aclError ret = aclrtDestroyStream(stream_);
ret = aclrtDestroyContext(context_);
ret = aclrtResetDevice(deviceId_);
ret = aclFinalize();

5.5 Application Build and Run

For details about how to build and run an application, see 8.3 Image Classification based on Caffe ResNet-50 (Including Image Decoding and Resizing) in the acl_dvpp_resnet50 sample.

Pay attention to the following points:

- **Model conversion:** See Restrictions and Parameters and Model Conversion Using ATC in ATC Tool Instructions.
- **Modification of the build script:**

  Obtain the build script `CMakeLists.txt` from the acl_dvpp_resnet50 sample and modify the following lines:
  
  - **include_directories:** Add the directories of the header files to be included.
    
    Example:
    
    ```
    include_directories(
        directoryPath1
        directoryPath2
    )
    ```

  - **link_directories:** Add the directories of the library files to be linked with.
    
    Example:
    
    ```
    link_directories(
        directoryPath3
        directoryPath4
    )
    ```

  - **add_executable:** Modify the name of an executable file (for example, `main`) and add the directory of the `.cpp` file.
    
    Example:
    
    ```
    add_executable(main
        directoryPath5
        directoryPath6)
    ```

  - **target_link_libraries:** Modify the name of the executable file (consistent with that of `add_executable`) and add the library files on which the executable file depends.
    
    Example:
    
    ```
    target_link_libraries(main
        ascendcl
        libName1
        libName2)
    ```

The library file is related to the header file that defines the API. The mapping is as follows:
### Table 5-1 Mapping between header files and library files

<table>
<thead>
<tr>
<th>Header File</th>
<th>Library File</th>
</tr>
</thead>
<tbody>
<tr>
<td>/acllib/include/acl/acl_base.h in the Acllib installation path</td>
<td>/acllib/lib64/stub/libascendcl.so in the ACLlib installation path</td>
</tr>
<tr>
<td>/acllib/include/acl/acl.h in the ACLlib installation path</td>
<td></td>
</tr>
<tr>
<td>/acllib/include/acl/acl_md1.h in the ACLlib installation path</td>
<td></td>
</tr>
<tr>
<td>/acllib/include/acl/acl_op.h in the Acllib installation path</td>
<td></td>
</tr>
<tr>
<td>/acllib/include/acl/acl_rt.h in the Acllib installation path</td>
<td></td>
</tr>
<tr>
<td>/acllib/include/acl/ops/acl_cblas.h in the Acllib installation path</td>
<td>/acllib/lib64/stub/libacl_cblas.so in the Acllib installation path</td>
</tr>
<tr>
<td>/acllib/include/acl/ops/acl_dvpp.h in the Acllib installation path</td>
<td>/acllib/lib64/stub/libacl_dvpp.so in the Acllib installation path</td>
</tr>
<tr>
<td>/acllib/include/acl/ops/acl fv.h in the Acllib installation path</td>
<td>/acllib/lib64/libacl_retr.so in the ACLlib installation path</td>
</tr>
<tr>
<td>(Currently, APIs in the header file cannot be referenced.)</td>
<td></td>
</tr>
</tbody>
</table>

**Note**

- The .so library file in the /acllib/lib64/stub directory of the Acllib installation path is used to build the code logic based on the ACL APIs without depending on any .so library of other components (such as Driver). Therefore, when using the cmake build command, set CMAKE_SKIP_RPATH to TRUE, indicating that the rpath, that is, the /acllib/lib64/stub directory in the Acllib installation path, is not added to the executable file generated during build.
- When the application runs with the build succeeded, you can configure the environment variables to link the application with the .so library file in the / acllib/lib64 directory in the Acllib installation path in the operating environment. During the running, the application is automatically linked with other .so library files that depend on other components.
- When building code logic based on the ACL APIs, link with the corresponding .so file based on the header file that is included. Linking unnecessary .so files may lead to compatibility problems during subsequent version upgrade.

Build options: Modify the name of the executable file (consistent with that of add_executable)

The following example indicates that the main file will be is installed in the $(CMAKE_INSTALL_PREFIX)/out directory. The path defined by the $(CMAKE_INSTALL_PREFIX) variable is relative to the path where the cmake command is executed.
set(CMAKE_INSTALL_PREFIX "../../../")
set(CMAKE_OUTPUT_DIR "out")

install(TARGETS main DESTINATION ${CMAKE_OUTPUT_DIR})

\[\text{NOTE}\]

For details about the cmake parameters, visit [https://cmake.org/cmake/help/latest/guide/tutorial/index.html](https://cmake.org/cmake/help/latest/guide/tutorial/index.html) and query the required version.

- Before running an executable file, upload the directories of the ACL initialization configuration file (\texttt{acl.json}), executable file, test image, and .om file to the same directory on the host.

If a null pointer is passed to the \texttt{aclInit} API during ACL initialization, the directory of the ACL initialization configuration file (\texttt{acl.json}) does not need to be uploaded to the host.
6.1 Stream Management

6.1.1 Basic Principles

A stream is a queue of tasks, with which the application manages the concurrency of tasks. The tasks in a stream are executed in its original order. Tasks in different streams are executed concurrently. A default stream is attached to a default context. If no stream is created explicitly, the default stream can be used. To pass the default stream to any API call, pass NULL directly.

6.1.2 Single-Thread, Single-Stream Scenario

```c
#include "acl/acl.h"
//......
// Explicitly create a stream.
aclrtStream stream;
aclrtCreateStream(&stream);

// Pass the stream argument to the task triggering call.
aclrtMemcpyAsync(devPtr, devSize, hostPtr, hostSize, ACL_MEMCPY_HOST_TO_DEVICE, stream);
// Call the aclrtSynchronizeStream interface to block application execution until all tasks in the stream are complete.
aclrtSynchronizeStream(stream);

// Explicitly destroy the stream after using the stream.
aclrtDestroyStream(stream);
//......
```
6.1.3 Multi-Thread, Multi-Stream Scenario

```c
#include "acl/acl.h"
//......
void runThread(aclrtStream stream) {
    int32_t deviceId =0;
    aclrtContext context;
    // If only one context is created, the thread uses this context by default.
    // If there are multiple contexts, aclrtSetCurrentContext should be called to set the context for the current thread.
    aclrtCreateContext(&context, deviceId);
    aclrtCreateStream(&stream);
    // Call the task triggering API.
    //....
    // Destroy resources.
    aclrtDestroyStream(stream);
    aclrtDestroyContext(context);
}

aclrtStream stream1;
aclrtStream stream2;
// Create two threads. Each thread corresponds to a stream.
std::thread t1(runThread, stream1);
std::thread t2(runThread, stream2);
// Explicitly call the join functions to ensure that the threads are ended.
t1.join();
t2.join();
```

6.2 Synchronous Wait

6.2.1 Basic Principles

The ACL supports the following synchronization mechanisms:

- Event synchronous wait: blocks application execution and waits until the event is complete by calling the `aclrtSynchronizeEvent` interface.
- Intra-stream task synchronous wait: blocks application execution and waits until all tasks in the specified stream are complete by calling the `aclrtSynchronizeStream` interface.
- Inter-stream task synchronous wait: blocks the running of a stream until the specified event is complete by calling the `aclrtStreamWaitEvent` interface. It is allowed for multiple streams to wait for a single event.
- Device synchronous wait: blocks application execution until the ongoing computation on the device is complete by calling the `aclrtSynchronizeDevice` interface. In the multi-device scenario, device of the current context is waited.

6.2.2 Event Synchronous Wait

```c
#include "acl/acl.h"
//......
// Create an event.
aclrtEvent event;
aclrtCreateEvent(&event);
// Create a stream.
```
6.2.3 Intra-Stream Task Synchronous Wait

```c
#include "acl/acl.h"

// Explicitly create a stream.
aclrtStream stream;
aclrtCreateStream(&stream);

// Pass the stream argument to the task triggering call.
aclrtMemcpyAsync(devPtr, devSize, hostPtr, hostSize, ACL_MEMCPY_HOST_TO_DEVICE, stream);

// Call aclrtSynchronizeStream to block application execution until all tasks in the specified stream are complete.
aclrtSynchronizeStream(stream);

// Explicitly destroy the stream after using the stream.
aclrtDestroyStream(stream);
```

6.2.4 Inter-Stream Task Synchronous Wait

```c
#include "acl/acl.h"

// Create an event.
aclrtEvent event;
aclrtCreateEvent(&event);

// Create two streams.
aclrtStream s1;
aclrtStream s2;
aclrtCreateStream(&s1);
aclrtCreateStream(&s2);

// Append an event to s1.
aclrtRecordEvent(event, s1);

// Block s2 until the event is complete, that is, the completion of s1.
// Wake up s2 for execution after s1 is complete.
aclrtStreamWaitEvent(s2, event);

// Explicitly destroy resources.
aclrtDestroyStream(s2);
aclrtDestroyStream(s1);
aclrtDestroyEvent(event);
```

6.2.5 Device Synchronous Wait

```c
aclrtSynchronizeDevice(); // Block application execution until the ongoing computation on the device is complete.
```
6.3 Data Copy

6.3.1 Basic Principles

- Data copy can be implemented in the following two modes:
  - Synchronous memory copy by calling `aclrtMemcpy`
  - Asynchronous memory copy by calling `aclrtMemcpyAsync`, and then call `aclrtSynchronizeStream` to implement intra-stream task synchronous wait
- Allocate memory on the host and the device in advance before memory copy.
  - Allocate the host memory by either calling the C++ interfaces `new` and `malloc` or the ACL interface `aclrtMallocHost`.
  - Allocate the device memory by calling the ACL interface `aclrtMalloc` interface. If data pre-processing (such as image decoding and resizing) is required, also call the `acldvppMalloc` to allocate the data pre-processing memory.
- When the executable file of an application can be executed on both the host and device, `aclrtGetRunMode` needs to be called to obtain the running mode of the software stack during programming, and the subsequent memory application interface calling logic is determined based on the running mode.
  a. If the executable file of an application is executed on the host, data transfer between the host and the device may be involved. In this case, the `aclrtMemcpy` interface (synchronous mode) or `aclrtMemcpy` interface (asynchronous mode) needs to be called to implement data transmission in memory replication mode.
  b. If the executable file of the application is executed on the device, data transfer between the host and the device is not involved.

6.3.2 Intra-Host Data Copy

A sample code is provided as follows.

```c
#include "acl/acl.h"

//......

//hostPtrA is the pointer to the destination memory address on the host, and hostSizeA indicates the size of the destination memory.
//hostPtrB is the pointer to the source memory address on the host, and count indicates the size of the source memory.
aclrtMemcpy(hostPtrA, hostSizeA, hostPtrB, count, ACL_MEMCPY_HOST_TO_HOST);

//......
```

6.3.3 Intra-Device Data Copy

A sample code is provided as follows.

```c
#include "acl/acl.h"

//......

// devPtrA is the pointer to the destination memory address on the device, and devSizeA indicates the size of the destination memory.
```
6.3.4 Host-to-Device Data Copy

The sample code is as follows:

```c
#include "acl/acl.h"

//......
// devPtrB is the pointer to the source memory address on the device, and count indicates the size of the source memory.
aclrtMemcpy(devPtrA, devSizeA, devPtrB, count, ACL_MEMCPYDEVICE_TODEVICE);

//......
```

6.3.5 Device-to-Host Data Copy

The sample code is as follows:

```c
#include "acl/acl.h"

//......
// hostPtr is the pointer to the destination memory address on the host, and count indicates the size of the destination memory.
//devPtrB is the pointer to the source memory address on the device, and count indicates the size of the source memory.
aclrtMemcpy(hostPtr, hostSize, devPtr, count, ACL_MEMCPYDEVICE_TOHOST);

//......
```

6.4 Data Preprocessing

6.4.1 General Description

- About asynchronous interfaces
  For the asynchronous interfaces described in this chapter, a successful interface call only indicates the success of the task delivery, regardless of the execution result. For dependent interfaces, you are advised to specify the same stream for multiple interfaces to ensure tasks execution sequence, because tasks in the same stream are executed in accordance with the interfaces calling sequence.

  When asynchronous APIs are called to decode, crop, and resize images, if tasks depend on each other, call `aclrtSynchronizeStream` to ensure that tasks in the stream are executed in order.

  After an asynchronous API is called, resources cannot be released immediately. You need to call the synchronous wait API (for example, `aclrtSynchronizeStream`) to confirm that the tasks on the device side haven been completed.

- About memory allocation and release
  If memory on the device is needed to store the input or output data before implementing the VPC, JPEGD, and JEPGE functions for media data
processing, \texttt{acldvppMalloc} can be called to allocate memory, and \texttt{acldvppFree} to free memory.

- The following media data processing functions are provided:
  - Vision preprocessing core (VPC): supports image cropping, resizing, overlaying, stitching, and format conversion. For details, see \texttt{6.4.2.1 Functions and Restrictions}.
  - JPEG decoder (JPEGD): decodes JPG, JPEG, JPG, and JPEG images into YUV images. For details, see \texttt{6.4.3.1 Functions and Restrictions}.
  - Video encoder (JPEGE): encodes YUV images into JPG images. For details, see \texttt{6.4.4.1 Functions and Restrictions}.
  - Video decoder (VDEC): decodes videos. For details, see \texttt{6.4.5.1 Functions and Restrictions}.
  - Video encoder (VENC): encodes videos. For details, see \texttt{6.4.6.1 Functions and Restrictions}.

- Except \texttt{acldvppMalloc} and \texttt{acldvppFree}, other APIs described in the media data processing chapter can be called only on the host side.

\section*{6.4.2 VPC}

\subsection*{6.4.2.1 Functions and Restrictions}

\textbf{Description}

The vision preprocessing core (VPC) provides the following functions:

- \textbf{Cropping}: crops one or more areas out of the input image.

- \textbf{Resizing}
  - VPC can process images with different resolutions:
    - Non-8K resizing: \texttt{widthStride} \( \in [32, 4096] \) and \texttt{heightStride} \( \in [6, 4096] \). The value range of \texttt{widthStride} varies according to the input image format. For details, see \texttt{Table 6-1}.
    - 8K resizing: \texttt{widthStride} \( \in (4096, 8192] \) or \texttt{heightStride} \( \in (4096, 8192] \)
  - Single-image cropping/resizing (supporting the uncompressed format) and single-image multi-ROI cropping/resizing (supporting the uncompressed format)
  - Other resizing modes, for example, original image resizing

- \textbf{Pasting}: crops an image out of an input image, resizes the cropped image, and pastes it in a specified area of the canvas. The canvas may be a blank image (when the output buffer allocated by the user is empty) or an existing image (when an image has been read into the output buffer allocated by the user). Note that the pasting concept here refers only to the case when the canvas is an existing image.

- \textbf{Stitching}: crops multiple images out of an input image, resizes the cropped images, and place them in a specified area of the output image.

- \textbf{Format conversion}
– Converts between RGB and YUV. For details about the input and output image formats, see Table 6-1.
– Converts a color image to a grayscale image. For the output image, only the data of the Y component is used.

Restrictions

● VPC has some restrictions on the input and output.
Table 6-1 Restrictions on VPC input and output

<table>
<thead>
<tr>
<th>VP C Input/Output</th>
<th>Resolution</th>
<th>Image Format</th>
<th>Memory</th>
<th>widthStride &amp; heightStride</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPC input</td>
<td>Non-8K resizing</td>
<td>The following enum values of <code>acldevppPixelFormat</code> are supported:</td>
<td></td>
<td>Calculate widthStride according to the image format.</td>
</tr>
<tr>
<td></td>
<td>YUV40 0 SP/ YUV42 0 SP/ YUV42 2 SP/ YUV44 4 SP: widthStride ∈ [32, 4096] and heightStride ∈ [6, 4096]</td>
<td>widthStride x heightStride x 3/2</td>
<td></td>
<td>- YUV40 0 SP and YUV42 0 SP: widthStride x heightStride x 3/2</td>
</tr>
<tr>
<td></td>
<td>YUV42 2 Packed: widthStride/2 ∈ [32, 4096] and heightStride ∈ [6, 4096]</td>
<td>widthStride x heightStride x 2</td>
<td></td>
<td>- YUV44 4 SP: widthStride x heightStride x 3</td>
</tr>
<tr>
<td></td>
<td>YUV44 4 Packed / RGB88 8: widthStride/3 ∈ [32, 4096] and</td>
<td>widthStride x</td>
<td></td>
<td>- YUV42 2 Packed: widthStride x</td>
</tr>
</tbody>
</table>

- YUV40 0 SP, YUV42 0 SP, YUV42 2 SP, and YUV44 4 SP: Round up the input width to the nearest multiple of 16.

- YUV42 2 Packed: Round up the input width to the nearest multiple of 16 and multiply the result by 2 (that is, each pixel...
<table>
<thead>
<tr>
<th>VP C Input/ Output</th>
<th>Resolution</th>
<th>Image Format</th>
<th>Memory</th>
<th>widthStride &amp; heightStride</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>XRGB8 888: widthStride/4 ∈ [32, 4096] and heightStride ∈ [6, 4096]</td>
<td></td>
<td>YUV44 4 Packed and RGB88 8: widthStride x height Stride</td>
<td>YUV44 4 Packed and RGB88 8: Round up the input width to the nearest multiple of 16 and multiply the result by 3 (that is, each pixel occupies 3 bytes).</td>
</tr>
<tr>
<td></td>
<td>8K resizing: widthStride ∈ (4096, 8192) or heightStride ∈ (4096, 8192)</td>
<td></td>
<td>XRGB8 888: widthStride x height Stride</td>
<td>XRGB8 888: Round up the input width to the nearest multiple of 16 and multiply the result by 4 (that...</td>
</tr>
</tbody>
</table>

Allocate device memory by calling acldvppMalloc and free device memory by calling acldvppFree.
<table>
<thead>
<tr>
<th>VP C Input/Output</th>
<th>Resolution</th>
<th>Image Format</th>
<th>Memory</th>
<th>widthStride &amp; heightStride</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>is, each pixel occupies 4 bytes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• <strong>heightStride</strong>: Round up the input height to the nearest multiple of 2.</td>
</tr>
<tr>
<td>VP C Input/Output</td>
<td>Resolution</td>
<td>Image Format</td>
<td>Memory</td>
<td>widthStride &amp; heightStride</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>VPC output</td>
<td>32 x 6 to 4096 x 4096</td>
<td>The following enum values of <code>acldvpPPixelFormat</code> are supported:</td>
<td>● The start address must be 16-byte aligned.</td>
<td>• <strong>widthStride</strong>: Round up the input width to the nearest multiple of 16.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1</code>, // 1,</td>
<td>● Calculate the output memory size according to the image format.</td>
<td>• <strong>heightStride</strong>: Round up the input height to the nearest multiple of 2.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>YUV420SP NV12 8bit</code></td>
<td>– YUV400 SP and YUV420 SP:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2</code>, // 2,</td>
<td><code>widthStride x heightStride x 3/2</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>YUV420SP NV21 8bit</code></td>
<td>– YUV422 SP:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>widthStride x heightStride x 2</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>– YUV444 SP:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>widthStride x heightStride x 3</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>– YUV422 Packed:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>widthStride x</code></td>
<td></td>
</tr>
<tr>
<td>VP C Input/ Output</td>
<td>Resolution</td>
<td>Image Format</td>
<td>Memory</td>
<td>widthStride &amp; heightStride</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------</td>
<td>--------------</td>
<td>--------</td>
<td>--------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>height Stride</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>YUV44</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Packed and RGB888</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>XRGB88</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>widthStride x height Stride</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Allocate device memory by calling acldvppMalloc and free device memory by calling acldvppFree.</td>
</tr>
</tbody>
</table>

- The resizing ratio of the width or height must be within the range of [1/32, 16].
- For `pasteArea`, the left-offset must be rounded up to the nearest multiple of 16.
VPC Function Diagram and Important Terms

**Figure 6-1** VPC function diagram (cropping+resizing+pasting)

**Figure 6-2** VPC function diagram (stitching)

**Table 6-2** Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>widthStride</td>
<td>Space occupied by each line of pixels in the buffer. <em>widthStride</em> of an RGB image is calculated differently from that of a YUV image. For details about the restrictions, see Table 6-1.</td>
</tr>
<tr>
<td>heightStride</td>
<td>Number of pixel lines in the buffer For details about the restrictions, see Table 6-1.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>------</td>
<td>------------</td>
</tr>
</tbody>
</table>
| Top/ Bottom/ Left/ Right offset | Configure the top offset, bottom offset, left offset, and right offset for the following purposes: (1) Specify the position of the cropArea or pasteArea. (2) Specify the width and height of the cropArea or pasteArea by using the following formulas: Right offset – Left offset + 1 = Width; Bottom offset – Top offset + 1 = Height.  
- Left offset: horizontal offset of points 1 and 3 in the cropArea or pasteArea relative to point 0  
- Right offset: horizontal offset of points 2 and 4 in the cropArea or pasteArea relative to point 0  
- Top offset: vertical offset of points 1 and 2 in the cropArea or pasteArea relative to point 0  
- Bottom offset: vertical offset of points 3 and 4 in the cropArea or pasteArea relative to point 0 |
| cropArea | Cropping area  
The minimum resolution is 10 x 6, and the maximum resolution is 4096 x 4096. |
| pasteArea | Pasting area in the canvas. The minimum resolution is 10 x 6, and the maximum resolution is 4096 x 4096.  
Restrictions:  
- For the pasteArea, the left offset and the top offset must be even numbers, and the right offset and the bottom offset must be odd numbers.  
- The cropArea cannot be larger than the input image, and the pasteArea cannot be larger than the canvas.  
- The pasteArea can be directly mapped on the leftmost side of the canvas, that is, the left offset of the output image is 0.  
- Up to 256 pasteAreas are allowed.  
- Round up the left offset of pasteArea relative to the output image to the nearest multiple of 16.  
- Round up the output pasteArea width to the nearest multiple of 16. Otherwise, padding is performed to meet the alignment requirement. |

**Performance Specifications**

- For **non-8K resizing**, the performance specifications in basic scenarios are as follows:
  
For 1080p images, if image data is copied from the host to the device, the maximum total frame rate is about 1000 fps due to the limited copy bandwidth.  
For 4K images, if the image data is copied from the host to the device, the maximum total frame rate is about 250 fps due to the copy bandwidth limit.
<table>
<thead>
<tr>
<th>Scenario</th>
<th>Total Frame Rate</th>
</tr>
</thead>
</table>
| - Input resolution: 1080p (1920 x 1080)  
- Output resolution: 1080p (1920 x 1080)  
- Input/Output format: YUV420 SP  
- n channels (n < 4, one channel corresponds to one thread) | n*360fps |
| - Input resolution: 1080p (1920 x 1080)  
- Output resolution: 1080p (1920 x 1080)  
- Input/Output format: YUV420 SP  
- n channels (n ≥ 4, one channel corresponds to one thread) | 1440fps |
| - Input resolution: 4K (3840 x 2160)  
- Output resolution: 4K (3840 x 2160)  
- Input/Output format: YUV420 SP  
- n channels (n < 4, one channel corresponds to one thread) | n*90fps |
| - Input resolution: 4K (3840 x 2160)  
- Output resolution: 4K (3840 x 2160)  
- Input/Output format: YUV420 SP  
- n channels (n ≥ 4, one channel corresponds to one thread) | 360fps |

- For **8K resizing**, the VPC performance is closely related to the output resolution. A higher output resolution indicates longer processing time and lower performance. The performance specifications in typical scenarios are as follows:

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Total Frame Rate</th>
</tr>
</thead>
</table>
| - Input resolution: 8K (7680 x 4320)  
- Output resolution: 1080p (1920 x 1080)  
- Input/Output format: YUV420 SP  
- n channels (n < 4, one channel corresponds to one thread) | n*4fps |
<table>
<thead>
<tr>
<th>Scenario</th>
<th>Total Frame Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Input resolution: 8K (7680 x 4320)</td>
<td>16fps</td>
</tr>
<tr>
<td>● Output resolution: 1080p (1920 x 1080)</td>
<td></td>
</tr>
<tr>
<td>● Input/Output format: YUV420 SP</td>
<td></td>
</tr>
<tr>
<td>● n channels (n ≥ 4, one channel corresponds to one thread)</td>
<td></td>
</tr>
<tr>
<td>● Input resolution: 8K (7680 x 4320)</td>
<td>n*1fps</td>
</tr>
<tr>
<td>● Output resolution: 4K (3840 x 2160)</td>
<td></td>
</tr>
<tr>
<td>● Input/Output format: YUV420 SP</td>
<td></td>
</tr>
<tr>
<td>● n channels (n &lt; 4, one channel corresponds to one thread)</td>
<td></td>
</tr>
<tr>
<td>● Input resolution: 8K (7680 x 4320)</td>
<td>4fps</td>
</tr>
<tr>
<td>● Output resolution: 4K (3840 x 2160)</td>
<td></td>
</tr>
<tr>
<td>● Input/Output format: YUV420 SP</td>
<td></td>
</tr>
<tr>
<td>● n channels (n ≥ 4, one channel corresponds to one thread)</td>
<td></td>
</tr>
</tbody>
</table>

### 6.4.2.2 Cropping

#### Basic Principles
- Create a channel for image data processing by calling `acldvppCreateChannel`.
- Call the asynchronous API `acldvppVpcCropAsync` to crop the image according to `cropArea` and loads the cropped image to the output memory for output. For asynchronous APIs, `aclrtSynchronizeStream` needs to be called to block the host until all tasks in the specified stream are complete.
- Resize the cropped image if the size is different from that of the canvas.

#### Sample Code
You can view the complete sample code in the `src/sample_process.cpp`, `src/dvpp_process.cpp`, and `src/utils.cpp` files of the `acl_vpc_jpege_resnet50` sample.

```cpp
#include "acl/acl.h"
#include "acl/ops/acldvpp.h"
//1. Initialize the ACL.
const char *aclConfigPath = "../src/acl.json";
aclError ret = aclInit(aclConfigPath);
//2. Allocate runtime resources, including devices, contexts, and streams.
ret = aclrtSetDevice(deviceId_);
```
ret = aclrtCreateContext(&context_, deviceId_);
ret = aclrtCreateStream(&stream_);

//3. Specify the crop area. cropArea_ is of type acldvppRoiConfig.
cropArea_ = acldvppCreateRoiConfig(550, 749, 480, 679);

//4. Create description of the data processing channel. dvppChannelDesc_ is of type acldvppChannelDesc.
dvppChannelDesc_ = acldvppCreateChannelDesc();

//5. Create a data processing channel.
acLError ret = acldvppCreateChannel(dvppChannelDesc_);

//6. Allocate input buffer cropInDevBuffer_. The buffer size cropInBufferSize is calculated based on the formula.
// If output buffer of other operations (such as decoding) is used as the input for image cropping, buffer
do not need to be allocated separately.
ret = acldvppMalloc(&cropInDevBuffer_, cropInBufferSize);

//7. Allocate output buffer cropOutBufferDev_. The buffer size cropOutBufferSize_ is calculated based on the formula.
ret = acldvppMalloc(&cropOutBufferDev_, cropOutBufferSize_);

//8. Create the description of the input image and set the attribute values. cropInputDesc_ is of type
acldvppPicDesc.
cropInputDesc_ = acldvppCreatePicDesc();
acldvppSetPicDescData(cropInputDesc_, cropInDevBuffer_);
acldvppSetPicDescFormat(cropInputDesc_, PIXEL_FORMAT_YUV_SEMIPLANAR_420);
acldvppSetPicDescWidth(cropInputDesc_, inputWidth_);
acldvppSetPicDescHeight(cropInputDesc_, inputHeight_);
acldvppSetPicDescWidthStride(cropInputDesc_, inputWidthStride);
acldvppSetPicDescHeightStride(cropInputDesc_, inputHeightStride);
acldvppSetPicDescSize(cropInputDesc_, cropInBufferSize);

//9. Create the description of the output image and set the attribute values. cropOutputDesc_ is of type
acldvppPicDesc.
// If the cropped output image is used as the input for model inference, the width and height of the output
image must meet the requirements of the model.
cropOutputDesc_ = acldvppCreatePicDesc();
acldvppSetPicDescData(cropOutputDesc_, cropOutBufferDev_);
acldvppSetPicDescFormat(cropOutputDesc_, PIXEL_FORMAT_YUV_SEMIPLANAR_420);
acldvppSetPicDescWidth(cropOutputDesc_, OutputWidth_);
acldvppSetPicDescHeight(cropOutputDesc_, OutputHeight_);
acldvppSetPicDescWidthStride(cropOutputDesc_, OutputWidthStride);
acldvppSetPicDescHeightStride(cropOutputDesc_, OutputHeightStride);
acldvppSetPicDescSize(cropOutputDesc_, cropOutBufferSize_);

//10. Perform asynchronous resizing and call aclrtSynchronizeStream to block the host until all tasks in the
specified stream are complete.
ret = acldvppVpcCropAsync(dvppChannelDesc_, cropInputDesc_,
cropOutputDesc_, cropArea_, stream_);
ret = aclrtSynchronizeStream(stream_);

//11. Release resources, including the description of the input and output image, input and output buffer,
description of the channel as well as the channel.
acldvppDestroyPicDesc(cropInputDesc_);
acldvppDestroyPicDesc(cropOutputDesc_);
(void)acldvppFree(cropInDevBuffer_);
(void)acldvppFree(cropOutBufferDev_);
adldvppDestroyChannel(dvppChannelDesc_);
(void)acldvppDestroyChannelDesc(dvppChannelDesc_);
dvppChannelDesc_ = nullptr;

//....
6.4.2.3 Cropping and Pasting

Basic Principles

- Create a channel for image data processing by calling `acldvppCreateChannel`.
- Call `acldvppVpcCropAndPasteAsync` asynchronous API to produce a cropped image according to `cropArea` and paste the cropped image to the `pasteArea` of the target image for output. For asynchronous APIs, `aclrtSynchronizeStream` needs to be called to block the host until all tasks in the specified stream are complete.
- Resize the image again if the size of the `cropArea` is inconsistent with that of the `pasteArea`.
- To paste the `pasteArea` on the canvas loaded to the output buffer, modify the code logic as follows: allocate the output buffer and then load a canvas to the allocated buffer.

Sample Code

You can view the complete sample code in the `src/sample_process.cpp`, `src/dvpp_process.cpp`, and `src/utils.cpp` files of the `acl_vpc_jpege_resnet50` sample.

```cpp
#include "acl/acl.h"
#include "acl/ops/acldvpp.h"

//1. Initialize the ACL.
const char *aclConfigPath = "../src/acl.json";
aclError ret = aclInit(aclConfigPath);

//2. Allocate runtime resources, including devices, contexts, and streams.
ret = aclrtSetDevice(deviceId_);
ret = aclrtCreateContext(&context_, deviceId_);
ret = aclrtCreateStream(&stream_);

//3. Specify the crop area and paste area. `cropArea_` and `pasteArea_` are of type `acldvppRoiConfig`.
cropArea_ = acldvppCreateRoiConfig(512, 711, 512, 711);
pasteArea_ = acldvppCreateRoiConfig(16, 215, 16, 215);

// 4. Create description of the data processing channel. `dvppChannelDesc_` is of type `acldvppChannelDesc`.
dvppChannelDesc_ = acldvppCreateChannelDesc();

//5. Create a data processing channel.
aclError ret = acldvppCreateChannel(dvppChannelDesc_);

// 6. Allocate input buffer `vpcInDevBuffer_`. The buffer size `vpcInBufferSize` is calculated based on the formula.
// If output buffer of other operations (such as decoding) is used as the input for image cropping and pasting, buffer does not need to be allocated separately.
//ret = acldvppMalloc(&vpcInDevBuffer_, vpcInBufferSize);

//7. Allocate output buffer `vpcOutBufferDev_`. The buffer size `vpcOutBufferSize_` is calculated based on the formula.
ret = acldvppMalloc(&vpcOutBufferDev_, vpcOutBufferSize_);

//8. Create the description of the input image and set the attribute values.
//In this sample, the decoding output buffer is used as the input of image cropping and pasting.
vpcInDesc_ = acldvppCreatePicDesc();
vpcInDesc_ = acldvppCreatePicDescData(vpcInDesc_, decodeOutBufferDev_);
acldvppSetPicDescData(vpcInDesc_, PIXEL_FORMAT_YUV_SEMIPLANAR_420);
acldvppSetPicDescWidth(vpcInDesc_, inputWidth_);
acldvppSetPicDescHeight(vpcInDesc_, inputHeight_);
acldvppSetPicDescWidthStride(vpcInDesc_, jpegOutWidthStride);
acldvppSetPicDescHeightStride(vpcInDesc_, jpegOutHeightStride);
```
//9. Create the description of the output image and set the attribute values.
// If the cropped and pasted output image is used as the input for model inference, the width and height of
// the output image must meet the requirements of the model.
// vpcOutputDesc_ is of type aclDvppPicDesc.
vpcOutputDesc_ = aclDvppCreatePicDesc();
acldvppSetPicDescData(vpcOutputDesc_, vpcOutBufferDev_);
acldvppSetPicDescFormat(vpcOutputDesc_, PIXEL_FORMAT_YUV_SEMIPLANAR_420);
acldvppSetPicDescWidth(vpcOutputDesc_, dvppOutWidth);
acldvppSetPicDescHeight(vpcOutputDesc_, dvppOutHeight);
acldvppSetPicDescWidthStride(vpcOutputDesc_, dvppOutWidthStride);
acldvppSetPicDescHeightStride(vpcOutputDesc_, dvppOutHeightStride);
acldvppSetPicDescSize(vpcOutputDesc_, vpcOutBufferSize_);

//10. Perform asynchronous resizing and call aclSynchronizeStream to block the host until all tasks in the
// specified stream are complete.
ret = aclDvppVpcCropAndPasteAsync(dvppChannelDesc_, vpcInputDesc_,
    vpcOutputDesc_, cropArea_, pasteArea_, stream_);
ret = aclSynchronizeStream(stream_);

//11. After decoding is complete, free resources, including the description of the input and output image,
// input and output buffer, description of the channel as well as the channel.
acldvppDestroyPicDesc(vpcInputDesc_);
acldvppDestroyPicDesc(vpcOutputDesc_);
// (void)acldvppFree(vpcInDevBuffer_);
// (void)acldvppFree(vpcOutBufferDev_);
acldvppDestroyChannel(dvppChannelDesc_);
// (void)acldvppDestroyChannelDesc(dvppChannelDesc_);
dvppChannelDesc_ = nullptr;

//....

6.4.2.4 Resizing

Basic Principles

- Create a channel for image data processing by calling aclDvppCreateChannel.
- Call the aclDvppVpcResizeAsync asynchronous API to resize the input image
  to the size of the output image. For asynchronous APIs, aclSynchronizeStream needs
to be called to block the host until all tasks in the specified stream are complete.
- Call aclDvppCreateResizeConfig to create resizing configuration data. The
  algorithm nearest neighbor interpolation is used by default, which cannot be
  modified.
- Calculate the buffer for storing the resized image based on the YUV420SP
  format as follows: Rounded-up width x Rounded-up height x 3/2.

Sample Code

You can view the complete sample code in the src/sample_process.cpp, src/
dvpp_process.cpp, and src/utils.cpp files of the acl_vpc_jpege_resnet50 sample.

#include "acl/acl.h"
#include "acl/ops/acl_dvpp.h"

//1. Initialize the ACL.
const char *aclConfigPath = "../src/acl.json";
aclError ret = aclInit(aclConfigPath);

//2. Allocate runtime resources, including devices, contexts, and streams.
ret = aclSetDevice(deviceId_);
ret = aclrtCreateContext(&context_, deviceId_);
ret = aclrtCreateStream(&stream_);

//3. Create resizing configuration data. The algorithm nearest neighbor interpolation is used by default, which cannot be modified.
// resizeConfig_ is of type acldvppResizeConfig
acldvppResizeConfig *resizeConfig_ = acldvppCreateResizeConfig();

// 4. Create description of the data processing channel. dvppChannelDesc_ is of type acldvppChannelDesc.
dvppChannelDesc_ = acldvppCreateChannelDesc();

//5. Create a data processing channel.
aclError ret = acldvppCreateChannel(dvppChannelDesc_);

// 6. Allocate input buffer resizeInDevBuffer_. The buffer size resizeInBufferSize is calculated based on the formula.
ret = acldvppMalloc(&resizeInDevBuffer_, resizeInBufferSize)

// 7. Allocate output buffer resizeOutBufferDev_. The buffer size resizeOutBufferSize_ is calculated based on the formula.
ret = acldvppMalloc(&resizeOutBufferDev_, resizeOutBufferSize_)

// 8. Create the description of the input image and set the attribute values.
// resizeInputDesc_ is of type acldvppPicDesc
resizeInputDesc_ = acldvppCreatePicDesc();

acldvppSetPicDescData(resizeInputDesc_, resizeInDevBuffer_);
acldvppSetPicDescFormat(resizeInputDesc_, PIXEL_FORMAT_YUV_SEMIPLANAR_420);
acldvppSetPicDescWidth(resizeInputDesc_, inputWidth_);
acldvppSetPicDescHeight(resizeInputDesc_, inputHeight_);
acldvppSetPicDescWidthStride(resizeInputDesc_, inputWidthStride);
acldvppSetPicDescHeightStride(resizeInputDesc_, inputHeightStride);
acldvppSetPicDescSize(resizeInputDesc_, resizeInBufferSize);

// 9. Create the description of the output image and set the attribute values.
// If the resized output image is used as the input for model inference, the width and height of the output image must meet the requirements of the model.
// resizeOutputDesc_ is of type acldvppPicDesc
resizeOutputDesc_ = acldvppCreatePicDesc();

acldvppSetPicDescData(resizeOutputDesc_, resizeOutBufferDev_);
acldvppSetPicDescFormat(resizeOutputDesc_, PIXEL_FORMAT_YUV_SEMIPLANAR_420);
acldvppSetPicDescWidth(resizeOutputDesc_, resizeOutputWidth_);
acldvppSetPicDescHeight(resizeOutputDesc_, resizeOutputHeight_);
acldvppSetPicDescWidthStride(resizeOutputDesc_, resizeOutputWidthStride);
acldvppSetPicDescHeightStride(resizeOutputDesc_, resizeOutputHeightStride);
acldvppSetPicDescSize(resizeOutputDesc_, resizeOutBufferSize_);

// 10. Perform asynchronous resizing and call aclrtSynchronizeStream to block the host until all tasks in the specified stream are complete.
ret = acldvppVpcResizeAsync(dvppChannelDesc_, resizeInputDesc_, resizeConfig_, stream_);
ret = aclrtSynchronizeStream(stream_);

// 11. After resizing is complete, free resources, including the description of the input and output image, as well as the input and output buffer.
acldvppDestroyPicDesc(resizeInputDesc_);
acldvppDestroyPicDesc(resizeOutputDesc_);
(void)acldvppFree(resizeInDevBuffer_);
(void)acldvppFree(resizeOutBufferDev_);

//....

6.4.3 JPEGD
6.4.3.1 Functions and Restrictions

Functions and Restrictions

The JPEG decoder (JPEGD) decodes .jpg, .jpeg, .JPG, and JPEG images. For the image decoding involving formats not supported by the hardware, software decoding is used.

- **About input**
  - Input resolution range:
    - 32 x 32 to 8192 x 8192
  - Input formats:
    - Only Huffman encoding is supported. The color space of the stream is YUV, and the subsample of the stream is 444, 422, 420, or 400.
    - Arithmetic coding is not supported.
    - The progressive JPEG format is not supported.
    - The JPEG2000 format is not supported.
  - Input memory:
    - The size of the input memory is the size of the input image.
    - The start address of the input memory must be 128-byte aligned. Allocate device memory by calling `acldvppMalloc` and free device memory by calling `acldvppFree`.

- **About output**
  - Output formats:
    After decoding for different source images, the JPEGD outputs images are in the following formats:
    - JPEG (YUV444 SP) -> YUV444 SP with V component before U component, YUV420 SP with V component before U component, or YUV420 SP with U component before V component
    - JPEG (YUV422 SP) -> YUV422 SP with V component before U component, YUV420 SP with V component before U component, or YUV420 SP with U component before V component
    - JPEG (YUV420 SP) -> YUV420 SP with V component before U component, or YUV420 SP with U component before V component
    - JPEG (YUV400) -> YUV420 SP with U and V components padded with 0x80
  - Output memory:
    - Calculate the output memory size according to the image format.
      - YUV420 SP: `widthStride x heightStride x 3/2`
      - YUV422 SP: `widthStride x heightStride x 2`
      - YUV444 SP: `widthStride x heightStride x 3`
The start address of the output memory must be 128-byte aligned. Allocate device memory by calling acldvppMalloc and free device memory by calling acldvppFree. For huge memory: Memory size = Output memory size + (n – 1) x AlignTo128(Output memory size + 8), where, n indicates the number of images.

- Output width and height:
  - **widthStride**: Round up the input width to the nearest multiple of 128.
  - **heightStride**: Round up the input height to the nearest multiple of 16.

- Hardware restrictions:
  - A maximum of four Huffman tables are supported, including two direct coefficient (DC) tables and two alternating coefficient (AC) tables.
  - A maximum of three quantization tables are supported.
  - Only 8-bit sampling is supported.
  - Only sequentially-encoded images can be decoded.
  - Only JPEG decoding based on discrete cosine transform (DCT) is supported.
  - Only one start of scan (SOS) flag is supported for image decoding.

- Software restrictions:
  - A maximum of three SOS flags are supported for image decoding.
  - Abnormal image decoding with insufficient minimum coded unit (MCU) data is supported.

### Performance Specifications

The performance specifications of the JPEGD are based on the hardware decoding performance. The JPEGD hardware decoding does not support the image decoding with three SOS flags. For image formats not supported by the hardware, software decoding is used. The software decoding performance for reference is 1080p x 1 channel at 15 fps.

1080p indicates a resolution of 1920 x 1080 pixels. 4K indicates a resolution of 3840 x 2160 pixels.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Total Frame Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-channel 1080p</td>
<td>128fps</td>
</tr>
<tr>
<td><em>n</em>-channel 1080p</td>
<td>256fps</td>
</tr>
<tr>
<td><em>n</em> ≥ 2</td>
<td></td>
</tr>
<tr>
<td>1-channel 4K</td>
<td>32fps</td>
</tr>
<tr>
<td><em>n</em>-channel 4K</td>
<td>64fps</td>
</tr>
<tr>
<td><em>n</em> ≥ 2</td>
<td></td>
</tr>
</tbody>
</table>
6.4.3.2 Image Decoding

Basic Principles

- Create a channel for image data processing by calling `aclDvppCreateChannel`.
- Call the `aclDvppJpegDecodeAsync` asynchronous API to decode JPG, JPEG, JPEG, and JPEG images. For asynchronous APIs, `aclrtSynchronizeStream` needs to be called to block the host until all tasks in the specified stream are complete.
- Call `aclDvppJpegPredictDecSize` to predict the size of the required output buffer for the decoded JPEG image based on the size of the JPEG image stored on the host.

Sample Code

You can view the complete sample code in the `src/sample_process.cpp`, `src/dvpp_process.cpp`, and `src/utils.cpp` files of the `acl_vpc_jpege_resnet50` sample.

```c
#include "acl/acl.h"
#include "acl/ops/acl_dvpp.h"

//1. Initialize the ACL.
const char *aclConfigPath = "../src/acl.json";
aclError ret = aciInit(aclConfigPath);

//2. Allocate runtime resources, including devices, contexts, and streams.
ret = aclrtSetDevice(deviceId_);
ret = aclrtCreateContext(&context_, deviceId_);
ret = aclrtCreateStream(&stream_);

//3. Load the image to the buffer. inDevBuffer_ indicates the buffer for storing the input image, and inDevBufferSize is the buffer size. The input buffer must be allocated in advance.
ret = aclDvppMalloc(&inDevBuffer_, inDevBufferSize);

//4. Create description of the data processing channel. dvppChannelDesc_ is of type aclDvppChannelDesc.
dvppChannelDesc_ = aclDvppCreateChannelDesc();

//5. Create a data processing channel.
aclError ret = aclDvppCreateChannel(dvppChannelDesc_);

//6. Allocate decoding output buffer decodeOutDevBuffer_. The buffer size decodeOutBufferSize is calculated based on the formula.
ret = aclDvppMalloc(&decodeOutDevBuffer_, decodeOutBufferSize);

//7. Create the description of the decoded image and set the attribute values.
//decodeOutputDesc is of type aclDvppPicDesc.
decodeOutputDesc_ = aclDvppCreatePicDesc();

//8. Perform asynchronous decoding and call aclrtSynchronizeStream to block the host until all tasks in the specified stream are complete.
ret = aclDvppJpegDecodeAsync(dvppChannelDesc_, inDevBuffer_, inDevBufferSize, decodeOutputDesc_, stream_1);
ret = aclrtSynchronizeStream(stream_1);

//9. Release resources, including the description of the output image, output buffer, description of the channel as well as the channel.
aclDvppDestroyPicDesc(decodeOutputDesc_);
```
6.4.4 JPEGE

6.4.4.1 Functions and Restrictions

Functions and Restrictions

The JPEG encoder (JPEGE) encodes YUV images into JPEG image files, for example, .jpg.

- About input
  - Input resolution range: 32 x 32 to 8192 x 8192
  - Input image formats:
    - YUV422 Packed (YUYV, YVYU, UYVY, VYUY)
    - YUV420 SP (NV12 and NV21)
  - Input width and height:
    - Round up widthStride to the nearest multiple of 16, for example, 128. For YUV422 Packed data, round up widthStride to the nearest multiple of 16, greater than twice the input width.
    - Round up heightStride to the input height or the nearest multiple of 16.
  - Input memory:
    - Calculate the input memory size according to the image format. YUV422 Packed: widthStride x heightStride YUV420 SP: widthStride x heightStride x 3/2
    - The start address of the input memory must be 128-byte aligned. Allocate device memory by calling acldvppMalloc and free device memory by calling acldvppFree.

- About output
  - The JPEG outputs image files in JPEG format, for example, .jpg.
  - Only Huffman encoding is supported. Arithmetic encoding and progressive encoding are not supported.
  - Output memory:
    - The size of the output memory is the size of the encoded image.
    - The start address of the output memory must be 128-byte aligned. Allocate device memory by calling acldvppMalloc and free device memory by calling acldvppFree.
### Performance Specifications

1080p indicates a resolution of 1920 x 1080 pixels. 4K indicates a resolution of 3840 x 2160 pixels.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Total Frame Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n)-channel 1080p ((n \geq 1))</td>
<td>64fps</td>
</tr>
<tr>
<td>(n)-channel 4K ((n \geq 1))</td>
<td>16fps</td>
</tr>
</tbody>
</table>

### 6.4.4.2 Image Encoding

#### Basic Principles

- Create a channel for image data processing by calling `acldvppCreateChannel`.
- Call the `acldvppJpegEncodeAsync` asynchronous API to encode a YUV image into a JPG image. For asynchronous APIs, `aclrtSynchronizeStream` needs to be called to block the host until all tasks in the specified stream are complete.
- Call `acldvppJpegPredictEncSize` to predict the size of the required output buffer for the encoded image based on the description of the input image and the image encoding configuration data.

#### Sample Code

You can view the complete sample code in the `src/sample_process.cpp`, `src/dvpp_process.cpp`, and `src/utils.cpp` files of the `acl_vpc_jpege_resnet50` sample.

```cpp
#include "acl/acl.h"
#include "acl/ops/acl_dvpp.h"

// 1. Initialize the ACL.
const char *aclConfigPath = "../src/acl.json";
acldvppCreateChannelDesc dvppChannelDesc_ = acldvppCreateChannelDesc();

// 2. Allocate runtime resources, including devices, contexts, and streams.
ret = aclSetDevice(deviceId_);
ret = aclrtCreateContext(&context_, deviceId_);
ret = aclrtCreateStream(&stream_);
ret = aclrtMallocHost(&inputHostBuff, PicBufferSize);
ret = acldvppMalloc(&inputDevBuff, PicBufferSize);
ret = aclrtMemcpy(inputDevBuff, PicBufferSize, inputHostBuff, PicBufferSize, ACL_MEMCPY_HOST_TO_DEVICE);
ret = aclrtMallocHost(&encodeOutBufferDev_, outBufferSize);
ret = aclrtMallocDev(&encodeOutBufferDev_, outBufferSize);
```

Copyright © Huawei Technologies Co., Ltd.
6.4.5 VDEC

6.4.5.1 Functions and Restrictions

The video decoder (VDEC) decodes videos. After VPC processing, the VDEC outputs YUV420 SP images (including NV12 and NV21).

- **About input**
  - Input resolution range:
    
    128 x 128 to 4096 x 4096
  - Input formats:
    
    - H.264 BP/MP/HP Level 5.1 YUV420 encoded streams
    - H.265 8-bit/10-bit Level 5.1 YUV420 encoded streams
  - Input memory:
    
    Allocate device memory by calling aclrtMalloc and free memory by calling aclrtFree. Alternatively, allocate device memory by calling acldvppMalloc and free memory by calling acldvppFree.

- **About output**
  - Output image formats:
- YUV420SP NV12
- YUV420SP NV21
- Output memory:
  - Calculate the output memory size according to the image format.
    YUV420 SP: \texttt{widthStride} \times \texttt{heightStride} \times \frac{3}{2}
  - The start address of the output memory must be 16-byte aligned.
    Allocate device memory by calling \texttt{aclrtMalloc} and free device memory by calling \texttt{aclrtFree}.
- Output width and height:
  - \texttt{widthStride}: Round up the input width to the nearest multiple of 16.
  - \texttt{heightStride}: Round up the input height to the nearest multiple of 2.
  - Bad frames or frame loss in the streams may cause VDEC frame loss.
  - The VDEC cannot decode the streams encoded in interlaced scanning mode.

6.4.5.2 Video Decoding

Basic Principles

1. Call \texttt{aclvdecCreateChannel} to create a channel for video stream data processing.
   - Perform the following steps before creating a channel for processing video stream data.
     i. Call \texttt{aclvdecCreateChannelDesc} to create channel description.
     ii. Call \texttt{aclvdecSetChannelDesc} series to set the decoding channel description attributes, including the decoding channel ID, thread, callback function, and video encoding protocol.
       1) The callback function needs to be created by the user in advance. It is used to obtain the decoding data after video decoding and free resources in a timely manner. For details about the callback function prototype, see \texttt{7.12.7.3 aclvdecCallback}.
        Call \texttt{acldvppGetPicDescRetCode} in the callback function to obtain the return code \texttt{retCode}. The value 0 indicates the success of decoding, while 1 indicates failure. If decoding fails, locate the fault based on the return code in the log. For details, see \texttt{Return Codes}.
        After the decoding is complete, you are advised to free the memory for storing the input streams and output images of the VDEC, and the corresponding video stream description and image description in the callback function in a timely manner.
        2) The user needs to create a thread in advance and customize a thread function. Calling \texttt{aclrtProcessReport} in the thread function triggers the callback function in 1.ii.1) after a specified period of time.
If `aclvdecSetChannelDescOutPicFormat` is not called to set the output format, images in YUV420 SP NV12 are output by default.

- The following APIs are encapsulated in `aclvdecCreateChannel` and do not need to be called separately:
  i. `aclrtCreateStream`: explicitly creates a stream. It is internally used for VDEC.
  ii. `aclrtSubscribeReport`: specifies a thread for processing the callback function in a stream. The callback function and thread are specified by calling the `aclvdecSetChannelDesc` series.

2. Decode a video stream into a YUV420SP image by calling `aclvdecSendFrame`.
   - Perform the following steps before decoding a video:
     - Call `acldvppCreateStreamDesc` to create the description of the input video stream, and call `acldvppSetStreamDesc` series to configure the input video, such as the memory address, memory size, and stream format.
     - Call `acldvppCreatePicDesc` to create the description of the output image, and call `acldvppSetPicDesc` series to configure the output image, such as the memory address, memory size, and image format.
   - `aclrtLaunchCallback` is encapsulated in the `aclvdecSendFrame` to add a callback function that needs to be executed on the host to the stream task queue. `aclrtLaunchCallback` does not need to be called separately.

3. Call `aclvdecDestroyChannel` to destroy a video processing channel.
   - The channel is destroyed only after the transmitted frames are decoded and the callback function is processed.
   - The following APIs are encapsulated in `aclvdecDestroyChannel` and do not need to be called separately:
     - `aclrtUnSubscribeReport`: unsubscribes a thread. (The callback function in the stream is no longer processed by the specified thread.)
     - `aclrtDestroyStream`: destroys a stream.
   - Call `aclvdecDestroyChannelDesc` to destroy the channel description after the channel is destroyed.

**Sample Code**

You can view the complete sample code in the `src` directory of the `acl_vdec_resnet50` sample.

```c
#include "acl/acl.h"
#include "acl/ops/acl_dvpp.h"

//1. Initialize the ACL.
const char *aclConfigPath = "../src/acl.json";
aclError ret = aclInit(aclConfigPath);

//2. Allocate runtime resources, including devices, contexts, and streams.
//ret = aclSetDevice(deviceId_);
//ret = aclrtCreateContext(&context_, deviceId_);
//ret = aclrtCreateStream(&stream_);
```
//3. Create a callback function.
void callback(acldvppStreamDesc *input, acldvppPicDesc *output, void *userdata)
{
    static int count = 1;
    if (output != nullptr) {
        //Obtain the VDEC output buffer, call the user-defined function WriteToFile to write output buffer
data to a file, and then call acldvppFree to free the output buffer.
        void *vdecOutBufferDev = acldvppGetPicDescData(output);
        if (vdecOutBufferDev != nullptr) {
            // 0: vdec success; others, vdec failed
            int retCode = acldvppGetPicDescRetCode(output);
            if (retCode == 0) {
                // process task: write file
                uint32_t size = acldvppGetPicDescSize(output);
                std::string fileNameSave = "outdir/image" + std::to_string(count);
                if (!Utils::WriteToFile(fileNameSave.c_str(), vdecOutBufferDev, size)) {
                    ERROR_LOG("write file failed.");
                } else {
                    ERROR_LOG("vdec decode frame failed.");
                }
            } else {
                ERROR_LOG("vdec decode frame failed.");
            }
            aclError ret = acldvppFree(vdecOutBufferDev);
            if (ret != ACL_ERROR_NONE) {
                ERROR_LOG("fail to free output pic desc data");
            }
        }
    }
    //Free data of the acldvppPicDesc type, which indicates the description data of the decoded output
image.
    aclError ret = acldvppDestroyPicDesc(output);
    if (ret != ACL_ERROR_NONE) {
        ERROR_LOG("fail to destroy output pic desc");
    }
}

// free output vdecOutBufferDev
aclError ret = acldvppFree(vdecOutBufferDev);
if (ret != ACL_ERROR_NONE) {
    ERROR_LOG("fail to free output pic desc data");
}

//Free data of the acldvppPicDesc type, which indicates the description data of the decoded output
image.
    aclError ret = acldvppDestroyPicDesc(output);
    if (ret != ACL_ERROR_NONE) {
        ERROR_LOG("fail to destroy output pic desc");
    }
}

// free input vdecInBufferDev and destroy stream desc
if (input != nullptr) {
    void *vdecInBufferDev = acldvppGetStreamDescData(input);
    if (vdecInBufferDev != nullptr) {
        aclError ret = acldvppFree(vdecInBufferDev);
        if (ret != ACL_ERROR_NONE) {
            ERROR_LOG("fail to free input stream desc data");
        }
    }
}

//Free data of type acldvppStreamDesc, which indicates the description data of the decoded input stream.
    aclError ret = acldvppDestroyStreamDesc(input);
    if (ret != ACL_ERROR_NONE) {
        ERROR_LOG("fail to destroy input stream desc");
    }

    INFO_LOG("success to callback %d.", count);
    count++;
}

//4. Create the description of the video stream processing channel, and set the attributes of the video
processing channel description. The callback function needs to be created in advance.
//vdecChannelDesc_ is of type aclvidecChannelDesc
    vdecChannelDesc_ = aclvidecCreateChannelDesc();
    // channelId: 0-15
    ret = aclvidecSetChannelDescChannelId(vdecChannelDesc_, 10);
    ret = aclvidecSetChannelDescThreadId(vdecChannelDesc_, threadId_);
    ret = aclvidecSetChannelDescCallback(vdecChannelDesc_, callback);
    //In the sample, the H265_MAIN_LEVEL video encoding protocol is used.
    ret = aclvidecSetChannelDescEnType(vdecChannelDesc_, static_cast<aclvppStreamFormat>(enType_));
    //In the sample, PIXEL_FORMAT_YUV_SEMIPLANAR_420 is used.
    ret = aclvidecSetChannelDescOutPixelFormat(vdecChannelDesc_, static_cast<aclvppPixelFormat>(format_));
// 5. Create a channel for processing video streams.
ret = aclvdecCreateChannel(vdecChannelDesc_);

// 6. Allocate device buffer dataDev to store the input video data for decoding.
// Copy image data from the host to the device by calling aclrtMemcpy. After data copy is complete, call
// aclrtFreeHost to free the host memory in a timely manner.
void *dataDev = nullptr;
auto aclRet = acldvvppMalloc(&dataDev, dataSize);
// dataHost indicates the buffer for storing the input video data on the host, and fileLen indicates the
// memory size.
aclRet = aclrtMemcpy(dataDev, dataSize, dataHost, fileLen, ACL_MEMCPY_HOST_TO_DEVICE);

// 7. Decode each of the 10 images in YUV420SP NV12 format.
int rest_len = 10;
int32_t count = 0;
while (rest_len > 0) {
    // 7.1 Create the description of the input video stream and set the stream attributes.
    streamInputDesc_ = acldvvppCreateStreamDesc();
    // inBufferDev_ indicates the buffer for storing the input video data on the device, and inBufferSize_ 
    // indicates the buffer size.
    ret = acldvvppSetStreamDescData(streamInputDesc_, inBufferDev_);
    ret = acldvvppSetStreamDescSize(streamInputDesc_, inBufferSize_);

    // 7.2 Allocate device buffer picOutBufferDev_ for storing the VDEC-decoded output data.
    ret = acldvvppMalloc(&picOutBufferDev_, size);

    // 7.3 Create the description of the output image and set the attributes of the image description.
    // picOutputDesc_ is of type acldvvppPicDesc.
    picOutputDesc_ = acldvvppCreatePicDesc();
    ret = acldvvppSetPicDescData(picOutputDesc_, picOutBufferDev_);
    ret = acldvvppSetPicDescSize(picOutputDesc_, size);
    ret = acldvvppSetPicDescFormat(picOutputDesc_, static_cast<acldvppPixelFormat>(format_));

    // 7.4 Decode video streams. The callback function is called after each decoded frame to write data to the
    // file, and free resources in a timely manner.
    ret = aclvdecSendFrame(vdecChannelDesc_, streamInputDesc_, picOutputDesc_, nullptr, nullptr);
    //.....
    ++count;
    rest_len = rest_len - 1;
    //.....
}

// 8. Destroy resources.
ret = aclvdecDestroyChannel(vdecChannelDesc_);
acldvppDestroyChannelDesc(vdecChannelDesc_);

//.....

Return Codes

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
<th>Possible Causes and Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>AICPU_DVPP_KERNEL_STATE_SUCCESS = 0</td>
<td>Decoding success</td>
<td>-</td>
</tr>
<tr>
<td>AICPU_DVPP_KERNEL_STATE_FAILED = 1</td>
<td>Other error</td>
<td>Rectify the fault based on the error logs, or contact Huawei technical support. For details about logs, see Log Reference.</td>
</tr>
<tr>
<td>Return Code</td>
<td>Description</td>
<td>Possible Causes and Solutions</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>--------------------------------------------------</td>
<td>-------------------------------------------------------------------</td>
</tr>
<tr>
<td>AICPU_DVPP_KERNEL_STATE_DVPP_ERROR = 2</td>
<td>Failed to call interfaces of other modules</td>
<td>Rectify the fault based on the error logs, or contact Huawei technical support. For details about logs, see Log Reference.</td>
</tr>
<tr>
<td>AICPU_DVPP_KERNEL_STATE_PARAM_INVALID = 3</td>
<td>Argument verification failure</td>
<td>Check whether the API arguments meet the API requirements.</td>
</tr>
<tr>
<td>AICPU_DVPP_KERNEL_STATE_OUTPUT_SIZE_INVALID</td>
<td>Output buffer size verification failure</td>
<td>Check whether the output buffer size meets the API requirements.</td>
</tr>
<tr>
<td>AICPU_DVPP_KERNEL_STATE_INTERNAL_ERROR = 5</td>
<td>System internal error</td>
<td>Rectify the fault based on the error logs, or contact Huawei technical support. For details about logs, see Log Reference.</td>
</tr>
<tr>
<td>AICPU_DVPP_KERNEL_STATE_QUEUE_FULL = 6</td>
<td>Full internal queue</td>
<td>Rectify the fault based on the error logs, or contact Huawei technical support. For details about logs, see Log Reference.</td>
</tr>
<tr>
<td>AICPU_DVPP_KERNEL_STATE_QUEUE_EMPTY = 7</td>
<td>Empty internal queue</td>
<td>Rectify the fault based on the error logs, or contact Huawei technical support. For details about logs, see Log Reference.</td>
</tr>
<tr>
<td>AICPU_DVPP_KERNEL_STATE_QUEUE_NOT_EXIST = 8</td>
<td>Nonexistent internal queue</td>
<td>Rectify the fault based on the error logs, or contact Huawei technical support. For details about logs, see Log Reference.</td>
</tr>
<tr>
<td>AICPU_DVPP_KERNEL_STATE_GET_CONTEXT_FAILED = 9</td>
<td>Internal context obtaining failure</td>
<td>Rectify the fault based on the error logs, or contact Huawei technical support. For details about logs, see Log Reference.</td>
</tr>
<tr>
<td>Return Code</td>
<td>Description</td>
<td>Possible Causes and Solutions</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>AICPU_DVPP_KERNEL_STATE_SUBMIT_EVENT_FAILED = 10</td>
<td>Internal event submission failure</td>
<td>Rectify the fault based on the error logs, or contact Huawei technical support. For details about logs, see Log Reference.</td>
</tr>
<tr>
<td>AICPU_DVPP_KERNEL_STATE_MEMORY_FAILED = 11</td>
<td>Internal memory allocation failure</td>
<td>Check whether the system has available memory.</td>
</tr>
<tr>
<td>AICPU_DVPP_KERNEL_STATE_SEND_NOTIFY_FAILED = 12</td>
<td>Internal notification sending failure</td>
<td>Rectify the fault based on the error logs, or contact Huawei technical support. For details about logs, see Log Reference.</td>
</tr>
<tr>
<td>AICPU_DVPP_KERNEL_STATE_VPC_OPERATE_FAILED = 13</td>
<td>Internal API operation failure</td>
<td>Rectify the fault based on the error logs, or contact Huawei technical support. For details about logs, see Log Reference.</td>
</tr>
<tr>
<td>ERR_INVALID_STATE = 0x10001</td>
<td>Abnormal VDEC decoder state</td>
<td>Rectify the fault based on the error logs, or contact Huawei technical support. For details about logs, see Log Reference.</td>
</tr>
<tr>
<td>ERR_HARDWARE = 0x10002</td>
<td>Hardware error, including decoder starts, execution, and stops</td>
<td>Rectify the fault based on the error logs, or contact Huawei technical support. For details about logs, see Log Reference.</td>
</tr>
<tr>
<td>ERR_SCD_CUT_FAIL = 0x10003</td>
<td>Abnormal video stream-to-frame cutting</td>
<td>Check whether the input video stream data is correct.</td>
</tr>
<tr>
<td>ERR_VDM_DECODE_FAIL = 0x10004</td>
<td>Single-frame decoding failure</td>
<td>Check whether the input video stream data is correct.</td>
</tr>
<tr>
<td>Return Code</td>
<td>Description</td>
<td>Possible Causes and Solutions</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>ERR_ALLOC_MEM_FAIL = 0x10005</td>
<td>Internal memory allocation failure</td>
<td>Check whether the system has available memory. <strong>If this error is ignored, no decoding result may be output with the continuously sent streams.</strong></td>
</tr>
<tr>
<td>ERR_ALLOC_DYNAMIC_MEM_FAIL = 0x10006</td>
<td>Out-of-range input video resolution and dynamic memory allocation failure</td>
<td>Check the resolution of the input video stream and whether the system has available memory. <strong>If this error is ignored, no decoding result may be output with the continuously sent streams.</strong></td>
</tr>
<tr>
<td>ERR_ALLOC_IN_OR_OUT_PORT_MEM_FAIL = 0x10007</td>
<td>Internal VDEC input and output buffer allocation failure</td>
<td>Check whether the system has available memory. <strong>If this error is ignored, no decoding result may be output with the continuously sent streams.</strong></td>
</tr>
<tr>
<td>ERR_BITSTREAM = 0x10008</td>
<td>Stream error (reserved)</td>
<td>-</td>
</tr>
<tr>
<td>ERR_VIDEO_FORMAT = 0x10009</td>
<td>Incorrect input video format</td>
<td>Check whether the input video format is H.264 or H.265.</td>
</tr>
<tr>
<td>ERR_IMAGE_FORMAT = 0x1000a</td>
<td>Incorrect output format</td>
<td>Check whether the output image format is NV12 or NV21.</td>
</tr>
<tr>
<td>ERR_CALLBACK = 0x1000b</td>
<td>Empty callback function</td>
<td>Check whether the callback function is empty.</td>
</tr>
<tr>
<td>ERR_INPUT_BUFFER = 0x1000c</td>
<td>Empty input buffer</td>
<td>Check whether the input buffer is empty.</td>
</tr>
<tr>
<td>ERR_INBUF_SIZE = 0x1000d</td>
<td>Buffer size error</td>
<td>Check whether the buffer size is less than or equal to 0.</td>
</tr>
<tr>
<td>ERR_THREAD&gt;Create_FB_FAIL = 0x1000e</td>
<td>Abnormal thread (thread for returning decoding result through the callback function)</td>
<td>Check whether the resources (such as threads and memory) in the system are available.</td>
</tr>
</tbody>
</table>
### 6.4.6 VENC

#### 6.4.6.1 Functions and Restrictions

**Functions and Restrictions**

The video encoder (VENC) encodes YUV420 SP NV12/NV21 8-bit image data into H.264/H.265 video streams. Multithreaded processes are not supported.

- About input

---

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
<th>Possible Causes and Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERR_CREATE_INSTANCE_FAIL = 0x1000f</td>
<td>Decoding instance creation failure</td>
<td>Rectify the fault based on the error logs, or contact Huawei technical support. For details about logs, see Log Reference.</td>
</tr>
<tr>
<td>ERR_INIT_DECODER_FAIL = 0x10010</td>
<td>Decoder initialization failure. For example, the number of decoding instances exceeds the maximum of 16.</td>
<td>Rectify the fault based on the error logs, or contact Huawei technical support. For details about logs, see Log Reference.</td>
</tr>
<tr>
<td>ERR_GET_CHANNEL_HANDLE_FAIL = 0x10011</td>
<td>Channel handle obtaining failure</td>
<td>Rectify the fault based on the error logs, or contact Huawei technical support. For details about logs, see Log Reference.</td>
</tr>
<tr>
<td>ERR_COMPONENT_SET_FAIL = 0x10012</td>
<td>Decoding instance setting failure</td>
<td>Check whether the input arguments are correct, for example, the input video format video_format and output frame format image_format.</td>
</tr>
<tr>
<td>ERR_COMPARE_NAME_FAIL = 0x10013</td>
<td>Decoding instance naming failure</td>
<td>Check whether the input arguments are correct, for example, the input video format video_format and output frame format image_format.</td>
</tr>
<tr>
<td>ERR_OTHER = 0x10014</td>
<td>Other error</td>
<td>Contact Huawei technical support.</td>
</tr>
</tbody>
</table>
- Input resolution range:
  128 x 128 to 1920 x 1920
- Input formats:
  YUV420 SP NV12/NV21-8bit
- Input memory:
  Allocate device memory by calling `acldvppMalloc` and free device memory by calling `acldvppFree`.

- About output
  - Output formats:
    H264 BP/MP/HP
    H.265 MP (slice streams only)
  - Output memory:
    The output memory is managed by the system.

### Performance Specifications

1080p indicates a resolution of 1920 x 1080 pixels.

<table>
<thead>
<tr>
<th>Scenario Example</th>
<th>Total Frame Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$-channel 1080p (one process corresponds to one channel)</td>
<td>30fps</td>
</tr>
</tbody>
</table>

### 6.4.6.2 Video Encoding

#### Basic Principles

1. Call `aclvencCreateChannel` to create a channel for video encoding.
   - Perform the following steps before creating a channel for video encoding.
     i. Call `aclvencCreateChannelDesc` to create the encoding channel description.
     ii. Call `aclvencSetChannelDesc series` to set the encoding channel description attributes, including the thread, callback function, video encoding protocol, and input image format.

2. The callback function needs to be created by the user in advance. It is used to obtain the encoding data after video encoding and free resources in a timely manner. For details about the callback function prototype, see 7.12.8.3 `aclvencCallback`.
   After the encoding is complete, you are advised to free the memory for storing the input images and the corresponding image description in the callback function in a timely manner.
   The output memory is managed by the system. Therefore, the output memory does not need to be freed by the user.

2) The user needs to create a thread in advance and customize a thread function. Calling `aclrtProcessReport` in the thread...
function triggers the callback function in 1.ii.1) after a specified period of time.

- The following APIs are encapsulated in aclvencCreateChannel and do not need to be called separately:
  
  i. aclrtCreateStream: explicitly creates a stream. It is internally used for VENC.

  ii. aclrtSubscribeReport: specifies a thread for processing the callback function in a stream. The callback function and thread are specified by calling the aclvencSetChannelDesc series.

2. Call aclvencSendFrame to encode YUV420SP images into H.264/H.265 video streams.

- Perform the following steps before encoding a video:

  ▪ Call acldvppCreatePicDesc to create the description of the input image, and call acldvppSetPicDesc series to configure the input image, such as the memory address, memory size, and image format.

  ▪ Call aclvencCreateFrameConfig to create the single-frame configuration data, and call aclvencSetFrameConfig series to configure whether to forcibly restart the I-frame interval or end the frame.

- aclrlaunchCallback is encapsulated in the aclvencSendFrame to add a callback function that needs to be executed on the host to the stream task queue. aclrlaunchCallback does not need to be called separately.

3. Call aclvencDestroyChannel to destroy a video processing channel.

- The channel is destroyed only after the transmitted frames are decoded and the callback function is processed.

- The following APIs are encapsulated in aclvencDestroyChannel and do not need to be called separately:

  ▪ aclrUnSubscribeReport: unsubscribes a thread. (The callback function in the stream is no longer processed by the specified thread.)

  ▪ aclrDestroyStream: destroys a stream.

- Call aclvencDestroyChannelDesc to destroy the channel description after the channel is destroyed.

6.5 Model Inference

6.5.1 Synchronous Inference (Batch Size = 1, Fixed Shape, Static AIPP, Single Model)

1. If Color Space Conversion (CSC), image normalization (by subtracting the mean value or multiplying a factor), image cropping (by specifying the crop start and cropping the image to the size required by the neural network) are involved, refer to the AIPP Configuration in ATC Tool Instructions before loading the model.
2. Before model inference, load model data from the file of the offline model adapted to the Ascend AI Processor to the memory, and create data of type aclmdlDataset to describe the output of the model. For details, see 5.4.1.4 Allocating Model Inference Resources.

3. After loading the model, perform model inference in synchronous mode. For details, see 5.4.4 Performing Model Inference (Batch Size = 1, Fixed Shape, Static AIPP, Single Model).

6.5.2 Multi-Model Inference

The basic procedure of multi-model inference is similar to that of single-model inference. For details, see 6.5.1 Synchronous Inference (Batch Size = 1, Fixed Shape, Static AIPP, Single Model).

The differences are as follows:

- The model loading API needs to be called separately for each model in multi-model scenarios.

  Model data can be loaded using the following APIs:

  - aclmdlLoadFromFile: loads offline model data from a file. The memory is managed by the system.
  - aclmdlLoadFromMem: loads offline model data from the memory. The memory is managed by the system.
  - aclmdlLoadFromFileWithMem: loads offline model data from a file. The memory (including working memory and weight memory) is managed by the user.
  - aclmdlLoadFromMemWithMem: loads offline model data from the memory. The memory (including working memory and weight memory) is managed by the user.

- The model inference API needs to be called separately for each model in multi-model scenarios.

  Call aclmdlExecute to execute synchronous model inference.

6.5.3 Batch Size Greater Than 1

The basic procedure of scenarios where the batch size is greater than 1 is similar to that of batch size equaling 1. For details, see 6.5.1 Synchronous Inference (Batch Size = 1, Fixed Shape, Static AIPP, Single Model).

In batchSize >1 scenarios, the following code logic needs to be added before model inference: if batchSize > 1 (for example, batchSize = 8), allocate memory on the device to store the multiple input data per batch for model inference. If the number of inputs does not meet the batchSize requirement, the remained data is used as the input for model inference.

The sample code is as follows (using batchSize = 8 as an example):

```c
uint32_t batchSize = 8;
uint32_t deviceNum = 1;
uint32_t deviceId = 0;

// Obtain the size of the first input of the model.
uint32_t modelInputSize = aclmdlGetInputSizeByIndex(modelDesc, 0);
// Obtain the input buffer size per batch.
uint32_t singleBuffSize = modelInputSize / batchSize;
```
// Define this variable to check whether the number of the accumulated inputs is greater than 8.
uint32_t cnt = 0;

// Define this variable to describe the offset when each file is loaded to the memory.
uint32_t pos = 0;

void* p_batchDst = NULL;
std::vector<std::string> inferFile_vec;

for (int i = 0; i < files.size(); ++i)
{
    // Allocate memory on the device once for every eight input files.
    if (cnt % batchSize == 0)
    {
        pos = 0;
        inferFile_vec.clear();
        // Allocate memory on the device.
        ret = aclrtMalloc(&p_batchDst, modelInputSize, ACL_MEM_MALLOC_NORMAL_ONLY);
    }

    // To-do: Load a file from a directory and calculate the file size fileSize.
    // Allocate memory on the host based on the file size to store file data.
    ret = aclrtMallocHost(&p_imgBuf, fileSize);

    // Copy file data from the host to the device.
    ret = aclrtMemcpy((uint8_t *)p_batchDst + pos, fileSize, p_imgBuf, fileSize, ACL_MEMCPY_HOST_TO_DEVICE);
    pos += fileSize;

    // Free memory on the host.
    aclrtFreeHost(p_imgBuf);

    // Save the i-th file to the vector and increase the value of cnt by 1.
    inferFile_vec.push_back(files[i]);
    cnt++;

    // Send the data per batch (batchSize = 8) for model inference.
    if (cnt % batchSize == 0)
    {
        // To-do: Create data of type aclmdlDataset and aclDataBuffer to describe the input and output
        // data of the model.
        // To-do: Call aclmdlExecute to start model inference.
        // To-do: Call aclrtFree to free memory on the device after the inference is complete.
    }
}

// If the number of inputs does not meet the batchSize requirement, the remained data is used as the input
// for model inference.
if (cnt % batchSize != 0)
{
    // To-do: Create data of type aclmdlDataset and aclDataBuffer to describe the input and output
    // data of the model.
    // To-do: Call aclmdlExecute to start model inference.
    // To-do: Call aclrtFree to free memory on the device after the inference is complete.
}

6.5.4 Dynamic Batch Size

Basic Principles

The API call sequence in the scenario with dynamic batch size is as follows:

1. Load the model. Successful model loading returns a model ID.
   Model data can be loaded using the following APIs:
   - aclmdlLoadFromFile: loads offline model data from a file. The memory
     is managed by the system.
- **aclmdlLoadFromMem**: loads offline model data from the memory. The memory is managed by the system.
- **aclmdlLoadFromFileWithMem**: loads offline model data from a file. The memory (including working memory and weight memory) is managed by the user.
- **aclmdlLoadFromMemWithMem**: loads offline model data from the memory. The memory (including working memory and weight memory) is managed by the user.

2. Create data of type **aclmdlDataset** to describe the input and output data of the model.
   
a. Call **aclCreateDataBuffer** to create data of type **aclDataBuffer** to store the buffer address and buffer size of the input and output data. The buffer needs to be allocated in advance by calling **aclrtMalloc**.

   Before allocating memory for the dynamic batch inputs, call **aclmdlGetInputIndexByNames** to obtain the index of the input dynamic batch size choices using the input name (names are fixed to **ACL_DYNAMIC_TENSOR_NAME**), and call **aclmdlGetInputSizeByIndex** to obtain the input size using the index of the input. Data in the memory does not need to be set after memory allocation (otherwise, services may be abnormal). After calling the APIs described in 3.b, the system automatically sets the data in the memory.

   **NOTE**

   Definition of the macro **ACL_DYNAMIC_TENSOR_NAME** is as follows:
   ```
   #define ACL_DYNAMIC_TENSOR_NAME "ascend_mbatch_shape_data"
   ```
   
b. Call **aclmdlCreateDataset** to create data of type **aclmdlDataset**, and call **aclmdlAddDatasetBuffer** to add data of type **aclDataBuffer** to data of type **aclmdlDataset**.

3. Set dynamic batch size choices after the model is successfully loaded before model execution.
   
a. Call **aclmdlGetInputIndexByNames** to obtain the index of the input dynamic batch size choices using the input name (names are fixed to **ACL_DYNAMIC_TENSOR_NAME**).

   b. Call **aclmdlSetDynamicBatchSize** to set the dynamic batch size.

   The batch size comes from the batch size choices specified by the **dynamic_batch_size** parameter during model conversion. For details about model conversion, see ATC Tool Instructions.

   You can also call the **aclmdlGetDynamicBatch** API to obtain the number of batch size choices as well as the batch size of each choice supported by the specified model.

4. Execute the model.

   Call **aclmdlExecute** to execute the model.

---

**NOTICE**

**aclmdlSetDynamicBatchSize** and **aclmdlSetDynamicHWSize** cannot be called for the same model at the same time.
**Sample Code**

```c
// 1. Load the model. Set dynamic batch size choices after the model is successfully loaded.
//......
// 2. Create data of type aclmdlDataset to describe the input data input_ and output data output_ of the model.
//......
//3. Call the customized function to set dynamic batch size choices.
int ModelSetDynamicInfo()
{
    size_t index;
    //2.1 Obtain the index of the input dynamic batch size choice. The input name is fixed to
    ACL_DYNAMIC_TENSOR_NAME.
    aclError ret = aclmdlGetInputIndexByName(modelDesc_, ACL_DYNAMIC_TENSOR_NAME, &index);
    //2.2 Set the batch size. modelId_ indicates the ID of a successfully loaded model, input_ indicates data of
    type aclmdlDataset, and index indicates the index of the input dynamic batch size choice.
    uint64_t batchSize = 8;
    ret = aclmdlSetDynamicBatchSize(modelId_, input_, index, batchSize);
    //......
}
//4. Customize a function and execute the model.
int ModelExecute(int index)
{
    aclError ret;
    //4.1 Call the customized function to set dynamic batch size choices.
    ret = ModelSetDynamicInfo();
    //4.2 Execute the model. modelId_ indicates the ID of a successfully loaded model, input_ indicates
    the input of a model, and output_ the output of a model.
    ret = aclmdlExecute(modelId_, input_, output_);
    //......
}
//5. Process the model inference result.
//TODO
```

### 6.5.5 Dynamic Image Size

**Basic Principles**

The procedure of calling key APIs in dynamic size scenarios is as follows:

1. **Load the model.** Successful model loading returns a model ID.
   - **Model data can be loaded using the following APIs:**
     - `aclmdlLoadFromFile`: loads offline model data from a file. The memory is managed by the system.
     - `aclmdlLoadFromMem`: loads offline model data from the memory. The memory is managed by the system.
     - `aclmdlLoadFromFileWithMem`: loads offline model data from a file. The memory (including working memory and weight memory) is managed by the user.
     - `aclmdlLoadFromMemWithMem`: loads offline model data from the memory. The memory (including working memory and weight memory) is managed by the user.

2. **Create data of type aclmdlDataset to describe the input and output data of the model.**
   - Call `aclCreateDataBuffer` to create data of type `aclDataBuffer` to store the buffer address and buffer size of the input and output data. The buffer needs to be allocated in advance by calling `aclrtMalloc`. 
Before allocating memory for the dynamic image size inputs, call `aclmdlGetInputIndexByName` to obtain the index of the input dynamic image sizes using the input name (names are fixed to `ACL_DYNAMIC_TENSOR_NAME`), and call `aclmdlGetInputSizeByIndex` to obtain the input size using the index of the input. Data in the memory does not need to be set after memory allocation (otherwise, services may be abnormal). After calling the APIs described in 3.b, the system automatically sets the data in the memory.

**NOTE**

Definition of the macro `ACL_DYNAMIC_TENSOR_NAME` is as follows:

```
#define ACL_DYNAMIC_TENSOR_NAME "ascend_mbatch_shape_data"
```

b. Call `aclmdlCreateDataset` to create data of type `aclmdlDataset`, and call `aclmdlAddDatasetBuffer` to add data of type `aclDataBuffer` to data of type `aclmdlDataset`.

3. Set the dynamic image size choices (width and height of the input image of the model) after the model is successfully loaded before model execution.

a. Obtain the index of the input dynamic image size choices using the input name (names are fixed to `ACL_DYNAMIC_TENSOR_NAME`)

b. Call `aclmdlSetDynamicHWSize` to set the dynamic image size.

   The image size comes from the image size choices specified by the `dynamic_image_size` parameter during model conversion. For details about model conversion, see ATC Tool Instructions.

   You can also call the `aclmdlGetDynamicHW` API to obtain the number of image size choices as well as the image size of each choice supported by the specified model.

4. Execute the model.
   Call `aclmdlExecute` to execute the model.

**NOTICE**

`aclmdlSetDynamicBatchSize` and `aclmdlSetDynamicHWSize` cannot be called for the same model at the same time.

---

### Sample Code

```c
// 1. Load the model. Set dynamic image size choices after the model is successfully loaded.
//......

// 2. Create data of type `aclmdlDataset` to describe the input data `input_` and output data `output_` of the model.
//......

// 3. Customize a function to set the dynamic image size choices.
int ModelSetDynamicInfo()
{
    size_t index;
    //2.1 Obtain the index of the input dynamic image size choice. The input name is fixed to
    //ACL_DYNAMIC_TENSOR_NAME
    aclError ret = aclmdlGetInputIndexByName(modelDesc_, ACL_DYNAMIC_TENSOR_NAME, &index);
    //2.2 Set the image size. `modelId_` indicates the ID of a successfully loaded model, `input_`
    //indicates data of type `aclmdlDataset`, and `index` indicates the index of the input dynamic image size choice.
    uint64_t height = 224;
    ```
6.5.6 Dynamic AIPP

Basic Principles

The procedure of calling key APIs in dynamic AIPP scenarios is as follows:

1. Load the model. Successful model loading returns a model ID.
   Model data can be loaded using the following APIs:
   - `aclmdlLoadFromFile`: loads offline model data from a file. The memory is managed by the system.
   - `aclmdlLoadFromMem`: loads offline model data from the memory. The memory is managed by the system.
   - `aclmdlLoadFromFileWithMem`: loads offline model data from a file. The memory (including working memory and weight memory) is managed by the user.
   - `aclmdlLoadFromMemWithMem`: loads offline model data from the memory. The memory (including working memory and weight memory) is managed by the user.

2. Create data of type `aclmdlDataset` to describe the input and output data of the model.
   a. Call `aclCreateDataBuffer` to create data of type `aclDataBuffer` to store the buffer address and buffer size of the input and output data. The buffer needs to be allocated in advance by calling `aclrtMalloc`.

   Before allocating memory for the dynamic AIPP inputs, call `aclmdlGetInputIndexByName` to obtain the index of the dynamic AIPP inputs using the input name (names are fixed to `ACL_DYNAMIC_AIPP_NAME`), and call `aclmdlGetInputSizeByIndex` to obtain the input size using the index of the input. Data in the memory does not need to be set after memory allocation (otherwise, services may be abnormal). After calling the APIs described in 3.c, the system automatically sets the data in the memory.

   **NOTE**

   Definition of the macro `ACL_DYNAMIC_AIPP_NAME` is as follows:
   ```
   #define ACL_DYNAMIC_AIPP_NAME "ascend_dynamic_aipp_data"
   ```
b. Call `aclmdlCreateDataset` to create data of type `aclmdlDataset`, and call `aclmdlAddDatasetBuffer` to add data of type `aclDataBuffer` to data of type `aclmdlDataset`.

3. Set **dynamic AIPP** parameters after the model is successfully loaded before model execution.
   a. Obtain the index of the dynamic AIPP inputs using the input name (names are fixed to `ACL_DYNAMIC_AIPP_NAME`.
   b. Call `aclmdlCreateAIPP` to create data of type `aclmdlAIPP`.
   c. Call the APIs provided in [7.15.17.2 Dynamic AIPP Parameters Setting](#) to set dynamic AIPP parameters as required.
   In dynamic AIPP scenarios, `aclmdlSetAIPPSrcImageSize` and `aclmdlSetAIPPSrcImageSize` must be called.
   d. Call `aclmdlSetInputAIPP` to set the dynamic AIPP data for model inference.
   e. Call `aclmdlDestroyAIPP` to destroy data of type `aclmdlAIPP` in a timely manner.

4. Execute the model.
   Call `aclmdlExecute` to execute the model.

**Sample Code**

```c
// 1. Load the model. Set dynamic AIPP attributes after the model is successfully loaded.
//......

// 2. Create data of type `aclmdlDataset` to describe the input data `input_` and output data `output_` of the model.
//......

// 3. Call the customized function to set the dynamic AIPP attributes
int ModelSetDynamicAIPP()
{
    //3.1 Obtain the index of the dynamic AIPP inputs.
    size_t index;
    aclError ret = aclmdlGetInputIndexByName(modelDesc_, ACL_DYNAMIC_AIPP_NAME, &index);

    //3.2 Set dynamic AIPP attributes.
    uint64_t batchNumber = 1;
    aclmdlAIPP *aippDynamicSet = aclmdlCreateAIPP(batchNumber);
    ret = aclmdlSetAIPPSrcImageSize(aippDynamicSet, 256, 224);
    ret = aclmdlSetAIPPSrcImageSize(aippDynamicSet, ACL_YUV420SP_U8);
    ret = aclmdlSetAIPPSrcImageSize(aippDynamicSet, 1, 256, 443, 0, 256, -86, -178, 256, 0, 350, 0, 0, 0, 128, 128);
    ret = aclmdlSetAIPPRbuSwapSwitch(aippDynamicSet, 0);
    ret = aclmdlSetAIPPDtcPixelMean(aippDynamicSet, 0, 0, 0, 0, 0);
    ret = aclmdlSetAIPPDtcPixelMin(aippDynamicSet, 0, 0, 0, 0, 0);
    ret = aclmdlSetAIPPDtcPixelVarReci(aippDynamicSet, 1, 1, 1, 1, 0);
    ret = aclmdlSetAIPPCscParams(aippDynamicSet, 1, 256, 443, 0, 256, -86, -178, 256, 0, 350, 0, 0, 0, 128, 128);
    ret = aclmdlSetAIPPCscParams(aippDynamicSet, 1, 256, 443, 0, 256, -86, -178, 256, 0, 350, 0, 0, 0, 128, 128);
    ret = aclmdlSetAIPPCscParams(aippDynamicSet, 1, 256, 443, 0, 256, -86, -178, 256, 0, 350, 0, 0, 0, 128, 128);
    ret = aclmdlSetAIPPCscParams(aippDynamicSet, 1, 256, 443, 0, 256, -86, -178, 256, 0, 350, 0, 0, 0, 128, 128);
    ret = aclmdlSetInputAIPP(modelId_, input_, index, aippDynamicSet);
    ret = aclmdlDestroyAIPP(aippDynamicSet);
    //......
}

//4. Customize a function and execute the model.
int ModelExecute(int index)
{
    aclError ret;
    //4.1 Call the customized function to set the dynamic AIPP attributes.
    ret = ModelSetDynamicAIPP();
    //......
}
```
6.6 Data Postprocessing

6.6.1 Post-processing Method for Target Classification Applications

For details, see 5.4.5 Post-Processing Data Using Single Operator and Returning Result to Host.

6.7 Single-Operator Calling

6.7.1 Built-in Operator GEMM Encapsulated into an ACL API

6.7.1.1 Basic Principles

Currently, the Gemm operator related to matrix-vector multiplication and matrix-matrix multiplication has been encapsulated into an ACL API. For details, see 7.11 CBLAS Interfaces.

The basic procedure of executing a single operator is as follows:

1. Initialize resources, including initializing the ACL, setting the loading directory of the single-operator model file, specifying the device for computation, and allocating memory on the device and host to store operator data.
   - Call aclInit to initialize the ACL.
   - Build the single operator definition file (.json) into an offline model supported by the Ascend AI processor (.om file) in advance by referring to Model Conversion Using ATC in ATC Tool Instructions.
   - A single-operator model file can be loaded using the following APIs:
     - aclopSetModelDir: sets the directory for loading the model file. The single-operator model file (.om file) is stored in the directory.
     - aclopLoad: loads the single-operator model data from the memory. The memory is managed by the user. Single-operator model data refers to the data that is loaded to the memory from the .om file. The .om file is built from a single operator.
   - Call aclrtSetDevice to specify the device for the computation.
   - Call aclrtCreateContext to explicitly create a context, and call aclrtCreateStream to explicitly create a stream.

   The default stream is used if no stream is created explicitly. The default stream is implicitly created in the aclrtSetDevice call. To pass the default stream to any API call, pass NULL directly.
2. Copy the operator input data from the host to the device.
   - Call `aclrtMemcpy` to implement synchronous memory copy.
   - Call `aclrtMemcpyAsync` to implement asynchronous memory copy.
3. Call the ACL API to execute the operator. This section uses the `aclblasGemmEx` API as an example.
4. Copy the output data of the operator from the device to the host.
   - Call `aclrtMemcpy` to implement synchronous memory copy.
   - Call `aclrtMemcpyAsync` to implement asynchronous memory copy.
5. Destroy streams and contexts and reset devices in sequence.
   - Call `aclrtDestroyStream` to destroy streams.
     If no stream is created explicitly and the default stream is used, `aclrtDestroyStream` does not need to be called.
   - Call `aclrtDestroyContext` to destroy contexts.
     If no context is created explicitly and the default context is used, `aclrtDestroyContext` does not need to be called.
   - Call `aclrtResetDevice` to reset devices.
6. Call `aclFinalize` to deinitialize the ACL.

6.7.1.2 Resource Initialization

After the API is called, add an exception handling branch and specify log printing of different levels (such as ERROR_LOG and INFO_LOG).

A sample code snippet is provided as follows. You can view the complete sample code in the `src/gemm_main.cpp` file of the `acl_execute_gemm` sample. (If no context is created explicitly, the default context is used.)

```cpp
#include "acl/acl.h"

//......

//1. Initialize the ACL.
//This path is relative to the directory of the executable file.
aclInit("test_data/config/acl.json");

//2. Set the directory of the single-operator model file.
//This directory is relative to the directory of the executable file. For example, if the executable file is stored in the run/out directory, the directory is run/out/op_models.
aclopSetModelDir("op_models");

//3. Specify the device for computation.
int deviceld = 0;
aclrtSetDevice(deviceld);

//4. Allocate memory on the device to store the input data of the operator.
//For this matrix-matrix multiplication sample, `sizeA_` indicates the size of matrix A, `sizeB_` the size of matrix B, and `sizeC_` the size of matrix C.
aclrtMalloc((void **) &devMatrixA_, sizeA_, ACL_MEM_MALLOC_NORMAL_ONLY)
aclrtMalloc((void **) &devMatrixB_, sizeB_, ACL_MEM_MALLOC_NORMAL_ONLY)
aclrtMalloc((void **) &devMatrixC_, sizeC_, ACL_MEM_MALLOC_NORMAL_ONLY)

//5. Allocate memory on the host to store the returned result of the operator.
//For this matrix-matrix multiplication sample, `m` indicates the number of rows of matrix A and matrix C, `n` indicates the number of columns of matrix B and matrix C, and `k` indicates the number of columns of matrix A and the number of rows of matrix B.
hostMatrixA_ = new(std::nothrow) aclFloat16[m_ * k_];
hostMatrixB_ = new(std::nothrow) aclFloat16[k_ * n_];
hostMatrixC_ = new(std::nothrow) aclFloat16[m_ * n_];
```
6.7.1.3 Copying Data to Device

After the API is called, add an exception handling branch and specify log printing of different levels (such as ERROR_LOG and INFO_LOG).

A sample code snippet is provided as follows. You can view the complete sample code in the src/gemm_main.cpp and src/gemm_runner.cpp files of the acl_execute_gemm sample.

```c
#include "acl/acl.h"
//......
//For this matrix-matrix multiplication sample, copy the data of matrix A and matrix B from the host to the device.
auto ret = aclrtMemcpy((void *) devMatrixA_, sizeA_, hostMatrixA_, sizeA_, ACL_MEMCPY_HOST_TO_DEVICE);
ret = aclrtMemcpy((void *) devMatrixB_, sizeB_, hostMatrixB_, sizeB_, ACL_MEMCPY_HOST_TO_DEVICE);
//......
```

6.7.1.4 Executing a Single Operator and Returning Result to the Host

After the API is called, add an exception handling branch and specify log printing of different levels (such as ERROR_LOG and INFO_LOG).

A sample code snippet is provided as follows. You can view the complete sample code in the src/gemm_main.cpp and src/gemm_runner.cpp files of the acl_execute_gemm sample.

```c
#include "acl/acl.h"
#include "acl/ops/acl_cblas.h"
//......
// Explicitly create a stream.
aclrtStream stream = nullptr;
aclrtCreateStream(&stream);
//In this example, aclblasGemmEx (asynchronous mode) is called to implement matrix-matrix multiplication.
aclblasGemmEx(ACL_TRANS_N, ACL_TRANS_N, ACL_TRANS_N, m_, n_, k_,
devAlpha_, devMatrixA_, k_, inputType_, devMatrixB_, n_, inputType_,
devBeta_, devMatrixC_, n_, outputType_, ACL_COMPUTE_HIGH_PRECISION,
stream);
// Call aclrtSynchronizeStream to block the host processing until all tasks in the specified streams are completed.
aclrtSynchronizeStream(stream);
// Copy the output data of the operator from the device to the host.
aclrtMemcpy(hostMatrixC_, sizeC_, devMatrixC_, sizeC_, ACL_MEMCPY_DEVICE_TO_HOST);
//To-do: Display the operator output data on the terminal screen and write the operator output data to a file.
//......
```

6.7.1.5 Releasing Runtime Resources and Deinitializing ACL

After the API is called, add an exception handling branch and specify log printing of different levels (such as ERROR_LOG and INFO_LOG).
A sample code snippet is provided as follows. You can view the complete sample code in the `src/gemm_main.cpp` and `src/gemm_runner.cpp` files of the `acl_execute_gemm` sample. If no context is created explicitly and the default context is used, the context does not need to be freed explicitly.

```c
#include "acl/acl.h"
//......
// Destroy the explicitly created streams.
(void) aclrtDestroyStream(stream);
// Reset devices.
(void) aclrtResetDevice(deviceld);
// Deinitialize the ACL.
aclFinalize();
//......
```

6.7.2 Built-in Operator Not Encapsulated into an ACL API

Basic Principles

The following is the basic procedure of executing a single operator. For details, see 5.4.5 Post-Processing Data Using Single Operator and Returning Result to Host.

1. Initialize resources, including initializing the ACL, setting the loading directory of the single-operator model file and specifying the device for computation.
   - Call `aiclInit` to initialize the ACL.
   - Build the single operator definition file (.json) into an offline model supported by the Ascend AI processor (.om file) in advance by referring to Model Conversion Using ATC in ATC Tool Instructions.
   - A single-operator model file can be loaded using the following APIs:
     - `aclopSetModelDir`: sets the directory for loading the model file. The single-operator model file (.om file) is stored in the directory.
     - `aclopLoad`: loads the single-operator model data from the memory. The memory is managed by the user. Single-operator model data refers to the data that is loaded to the memory from the .om file. The .om file is built from a single operator.
   - Call `aclrtSetDevice` to specify the device for the computation.
   - Call `aclrtCreateContext` to explicitly create a context, and call `aclrtCreateStream` to explicitly create a stream.
     - The default stream is used if no stream is created explicitly. The default stream is implicitly created in the `aclrtSetDevice` call. To pass the default stream to any API call, pass `NULL` directly.

2. Copy the operator input data from the host to the device.
   - Call `aclrtMemcpy` to implement synchronous memory copy.
   - Call `aclrtMemcpyAsync` to implement asynchronous memory copy.

3. Execute the single operator.
   - A single operator can be executed in the following two modes:
     - Construct the operator description information (such as the input and output tensor description and operator attributes), allocate memory for
storing the input and output data of the operator, and call \texttt{aclopExecute} to load and execute the operator.

In this mode, the system matches the model in the memory based on the operator description in every \texttt{aclopExecute} call to execute the operator.

- Construct the operator description information (such as the input and output tensor description and operator attributes), allocate memory for storing the input and output data of the operator, call \texttt{aclopCreateHandle} to create a \texttt{Handle}, and call \texttt{aclopExecWithHandle} to load and execute an operator.

In this mode, when \texttt{aclopCreateHandle} is called, the system matches the model in the memory based on the operator description, which is cached in the \texttt{Handle}. The \texttt{Handle} improves the efficiency for scenarios where the same operator is executed for multiples times with the \texttt{aclopExecWithHandle} call. Call \texttt{aclopDestroyHandle} to destroy the handle when it is no longer needed.

4. Copy the output data of the operator from the device to the host (memory on the host needs to be allocated in advance).
   - Call \texttt{aclrtMemcpy} to implement synchronous memory copy.
   - Call \texttt{aclrtMemcpyAsync} to implement asynchronous memory copy.

5. Destroy streams and contexts and reset devices in sequence.
   - Call \texttt{aclrtDestroyStream} to destroy streams.
     If no stream is created explicitly and the default stream is used, \texttt{aclrtDestroyStream} does not need to be called.
   - Call \texttt{aclrtDestroyContext} to destroy contexts.
     If no context is created explicitly and the default context is used, \texttt{aclrtDestroyContext} does not need to be called.
   - Call \texttt{aclrtResetDevice} to reset devices.

6. Call \texttt{aclFinalize} to deinitialize the ACL.

Sample Code

For details, see 5.4.5 Post-Processing Data Using Single Operator and Returning Result to Host.

6.7.3 Custom Operator

6.7.3.1 Loading and Executing a Fixed Shape Operator

1. For details about developing custom operators, see Operator Development Flow in \textit{TBE Custom Operator Development Guide}.

2. For details about the calling process of a developed single operator, see 6.7.2 Built-in Operator Not Encapsulated into an ACL API.

6.7.3.2 Loading and Executing a Dynamic-Shape Operator

Before loading and executing a dynamic-shape operator, develop the custom operator and generate the corresponding binary file by referring to "TIK Operator with Dynamic Shape" in \textit{TBE Custom Operator Development Guide}. 
Basic Principles

The procedure of loading and executing a dynamic-shape operator is as follows:

1. Initialize resources, including initializing the ACL, setting the loading directory of the single-operator model file and specifying the device for computation.
   - Call `aclInit` to initialize the ACL.
   - Call the ACL APIs to register the custom operator to be built.
     ▪ Call `aclopRegisterCompileFunc` to register the operator selector (that is, selecting the tiling policy function). Different tiling strategies are adopted for different shapes when executing the operator.
       The operator selector needs to be defined and implemented by the user in advance.
       ○ Function prototype
         ```c
         typedef aclError (*aclopCompileFunc)(int numInputs, const aclTensorDesc *const inputDesc[], int numOutputs, const aclTensorDesc *const outputDesc[], const aclopAttr *opAttr, aclopKernelDesc *aclopKernelDesc);
         ```
       ○ Function implementation
         You can write code logic to select a tiling policy and generate tiling parameters, and call `aclopSetKernelArgs` to set tiling arguments and number of blocks for concurrent execution.
     ▪ Call `aclopCreateKernel` to register the operator to the system for code implementation when executing the operator.
       - Call `aclrtSetDevice` to specify the device for the computation.
       - Call `aclrtCreateContext` to explicitly create a context, and call `aclrtCreateStream` to explicitly create a stream.
         The default stream is used if no stream is created explicitly. The default stream is implicitly created in the `aclrtSetDevice` call. To pass the default stream to any API call, pass `NULL` directly.

2. Construct the operator description (such as the input and output tensor description and operator attributes) and allocate memory for storing the input and output data of the operator.

3. Copy the operator input data from the host to the device.
   - Call `aclrtMemcpy` to implement synchronous memory copy. The memory needs to be freed in a timely manner after being used.
   - Call `aclrtMemcpyAsync` to implement asynchronous memory copy. The memory needs to be freed in a timely manner after being used.

4. Build a single operator.
   Call `aclopUpdateParams` to build the operator and trigger the calling logic of the operator selector.

5. Execute the single operator.
   Call `aclopExecute` to load and execute the operator.

6. Copy the output data of the operator from the device to the host (memory on the host needs to be allocated in advance).
   - Call `aclrtMemcpy` to implement synchronous memory copy. The memory needs to be freed in a timely manner after being used.
7. Destroy streams and contexts and reset devices in sequence.
   - Call aclrtDestroyStream to destroy streams.
     If no stream is created explicitly and the default stream is used, aclrtDestroyStream does not need to be called.
   - Call aclrtDestroyContext to destroy contexts.
     If no context is created explicitly and the default context is used, aclrtDestroyContext does not need to be called.
   - Call aclrtResetDevice to reset devices.

8. Call aclFinalize to deinitialize the ACL.

Sample Code

A sample code snippet is provided as follows. You can view the complete sample code in the ACLlib installation path/acllib/sample/acl_execute_op/acl_execute_batchnorm/src directory.

```c
#include "acl/acl.h"

// Initialize resources.
// This path is relative to the directory of the executable file.
aclopRegisterCompileFunc("BatchNorm", SelectAclopBatchNorm);
// Build the .o file of the operator kernel in advance and call the user-defined function to load the file to the memory buffer. length indicates the memory size. If there are multiple .o files, this API needs to be called for multiple times.
aclopCreateKernel("BatchNorm", "tiling_mode_1__kernel0", "tiling_mode_1__kernel0", buffer, length, ACL_ENGINE_AICORE, Deallocator);

// Customize the BatchNormTest(n, c, h, w) function and perform the following operations:

// Construct the input and output tensor descriptions of the BatchNorm operator as well as the descriptions of the input and output tensor, and allocate memory for storing the input and output data of the operator.
ac TensorDesc *input_desc[3];
ac TensorDesc *output_desc[1];
input_desc[0] = acCreateTensorDesc(ACL_FLOAT16, 4, shape_input, ACL_FORMAT_NCHW);
input_desc[1] = acCreateTensorDesc(ACL_FLOAT16, 1, shape_gamma, ACL_FORMAT_ND);
input_desc[2] = acCreateTensorDesc(ACL_FLOAT16, 1, shape_beta, ACL_FORMAT_ND);
output_desc[0] = acCreateTensorDesc(ACL_FLOAT16, 4, shape_out, ACL_FORMAT_NCHW);

for (int i = 0; i < n * c * h * w; ++i) {
    input[i] = aclFloatToFloat16(1.0f);
}

for (int i = 0; i < c; ++i) {
    gamma[i] = aclFloatToFloat16(0.5f);
    beta[i] = aclFloatToFloat16(0.1f);
}

devInput = aclrtMalloc(&devInput, size_input, ACL_MEM_MALLOC_NORMAL_ONLY);
devInput_gamma = aclrtMalloc(&devInput_gamma, size_gamma, ACL_MEM_MALLOC_NORMAL_ONLY);
devInput_beta = aclrtMalloc(&devInput_beta, size_beta, ACL_MEM_MALLOC_NORMAL_ONLY);
devOutput = aclrtMalloc(&devOutput, size_output, ACL_MEM_MALLOC_NORMAL_ONLY);

// Copy the operator input data from the host to the device.
devInput = aclrtMemcpy(devInput, size_input, input, size_input, ACL_MEMCPY_HOSTToDevice);
devInput_gamma = aclrtMemcpy(devInput_gamma, size_gamma, gamma, size_gamma, ACL_MEMCPY_HOSTToDevice);
devInput_beta = aclrtMemcpy(devInput_beta, size_beta, beta, size_beta, ACL_MEMCPY_HOSTToDevice);
```
// 4. Call aclopUpdateParams to build the operator.
aclopUpdateParams("BatchNorm", 3, input_desc, 1, output_desc, nullptr, ACL_ENGINE_AICORE,
ACL_COMPILE_UNREGISTERED, nullptr));

// 5. Call aclopExecute to load and execute the operator.
aclopExecute("BatchNorm", 3, input_desc, inputs, 1, output_desc, outputs, nullptr, stream);

// Customize the BatchNormTest(n, c, h, w) function and perform the preceding operations.

// 6. Copy the output data of the operator from the device to the host (memory on the host needs to be
allocated in advance).
acirtMemcpy(output, size_output, devOutput, size_output, ACL_MEMCPY_DEVICE_TO_HOST);

// 7. Free the resources in sequence.
// 7.1 Free the input and output tensor description.
for (auto desc : input_desc) {
    aclDestroyTensorDesc(desc);
}
for (auto desc : output_desc) {
    aclDestroyTensorDesc(desc);
}
// 7.2 Free the memory on the host.
delete[]input;
delete[]gamma;
delete[]beta;
delete[]output;
// 7.3 Free the memory on the device.
acirtFree(devInput);
acirtFree(devInput_gamma);
acirtFree(devInput_beta);
acirtFree(devOutput);
// 7.4 Destroy streams and contexts and reset devices in sequence. Skip this step if no stream or context is
created explicitly.
acirtDestroyStream(stream);
acirtDestroyContext(context);
acirtResetDevice(deviceId);
aclFinalize();
7 ACL API Reference

7.1 Introduction

7.1.1 ACL Overview
Ascend Computing Language (ACL) provides a collection of C++ APIs for users to develop deep neural network applications for object recognition and image classification, ranging from device management, context management, streams management, memory management, to model loading and execution, operator loading and execution, and media data processing. Users can call ACL APIs directly or through a third-party framework to utilize the computing capability of the Ascend AI Processor, or encapsulate the ACL APIs to implement a third-party
library, to utilize the execution and resources management capabilities of the Ascend AI Processor.

When running an application, ACL calls APIs provided by the Graph Engine Executor (GE) to load and run models and operators, and calls Runtime APIs to manage devices, contexts, streams, and memory.

The Al Core and AI CPU, as the hardware computing capability basis of the Ascend AI Processor, perform matrix-related computation of neural networks, general computation and execution control of control operators, scalars, and vectors, as well as image and video data preprocessing. AI Core and AI CPU guarantee the execution of the deep neural network (NN) computing.

**Figure 7-1 Logical architecture**

![Logical architecture diagram](image)
7.1.2 Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous/Asynchronous</td>
<td>Scheduling is classified into the synchronous and asynchronous modes in this document from the perspective of the caller and executor. In the current scenario, if the API calling result is returned immediately without waiting for the device to complete the execution, the host scheduling is in asynchronous mode. If the API calling result is not returned until the device completes the execution, the host scheduling is in synchronous mode.</td>
</tr>
<tr>
<td>Process/Thread</td>
<td>Unless otherwise specified, the processes and threads mentioned in this document refer to the processes and threads on the host.</td>
</tr>
<tr>
<td>Host</td>
<td>The host side refers to the x86 or Arm server connected to the device side. The host side utilizes the NN computing capability provided by the device side to implement services.</td>
</tr>
<tr>
<td>Device</td>
<td>The device side refers to the hardware device that provides the host with the NN computing capability over the PCIe interface.</td>
</tr>
<tr>
<td>Context</td>
<td>As a container, the context manages the life cycle its objects (including streams, events, and device memory). Streams and events in different contexts are isolated, and cannot be executed synchronously.</td>
</tr>
<tr>
<td></td>
<td>There are two types of contexts:</td>
</tr>
<tr>
<td></td>
<td>● Default context: a default context created implicitly upon the aclrtSetDevice call that specifies the device for computation. Each device corresponds to a default context. A default context cannot be released by calling aclrtDestroyContext.</td>
</tr>
<tr>
<td></td>
<td>● Explicitly created context (recommended): a context created explicitly upon the aclrtCreateContext call in a process or thread</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Stream | Streams preserve the execution order of a stack of asynchronous operations that are executed on the device in its original order. Stream-based kernel execution and data transfer can implement the parallelism of computing on both the host and device sides, and data transfer between the host and device sides. Streams come in two types:  
  - Default stream: a default stream created implicitly upon the aclrtSetDevice call that specifies the device for computation. Each device corresponds to a default stream. A default stream cannot be released by calling aclrtDestroyStream.  
  - Explicitly created stream (recommended): a stream created explicitly upon the aclrtCreateStream call in a process or thread. |
| Event  | Supports tasks synchronization between streams by calling ACL APIs, including tasks between the host and device, and tasks between the devices. For example, if task of stream2 needs to be executed after task in stream1 is complete, you can create an event and insert it to stream1. |
| AIPP   | The AIPP module is introduced for image pre-processing including Color Space Conversion (CSC), image normalization (by subtracting the mean value or multiplying a factor), image cropping (by specifying the crop start and cropping the image to the size required by the neural network), and much more. AIPP supports static and dynamic modes. However, the two modes are mutually exclusive.  
  - Static AIPP: During model conversion, set the AIPP mode to static and set the AIPP parameters. After the model is generated, the AIPP parameter values are saved in the offline model (*.om). The same AIPP parameter configurations are used in each model inference phase. In static AIPP mode, batches share the same set of AIPP parameters.  
  - Dynamic AIPP: During model conversion, specify the AIPP mode to dynamic, and set different sets of dynamic AIPP parameters as required. In this way, different sets of parameters can be used for model inference. For details about the interface for setting dynamic AIPP parameters, see 7.15.17.2 Dynamic AIPP Parameters Setting. In dynamic AIPP mode, batches can use different sets of AIPP parameters. |
### 7.2 Initialization and Deinitialization

#### 7.2.1 aclInit

**Description**

Initializes ACL function. It is a synchronous interface.

**Restrictions**

- `aclInit` can be called only once in a process.
- `aclInit` must be called before application development using ACL interface. Otherwise, an error may occur during the initialization of internal system resources, causing other service exceptions.

**Prototype**

```c
aclError aclInit(const char *configPath)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>configPath</td>
<td>Input</td>
<td>Path of the configuration file, including the file name. The configuration file is in JSON format. Currently, dump data is configurable. For details, see <a href="#">Configuration File Example</a>. For details about the configuration description, see Preparing Data &gt; Preparing Dump Data for an Offline Model in Model Accuracy Analyzer Instructions (CLI). If no configuration information is required, pass <code>nullptr</code> to the <code>aclInit</code> call.</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

Configuration File Example

Take the Caffe ResNet-50 model as an example. To compare the output results of an operator layer in the offline model adapted to the Ascend AI Processor converted from Caffe ResNet-50 and those converted from the Caffe ResNet-50 model, configure the configuration file as follows:

```json
{
  "dump":{
    "dump_list":[
    {
      "model_name":"ResNet-50",
      "layer":{
        "conv1conv1_relu",
        "res2a_branch2ares2a_branch2a_relu",
        "res2a_branch1",
        "pool1"
      }
    },
    "dump_path":"/MyApp20/dump",
    "dump_mode":"output"
  }
}
```

7.2.2 aclFinalize

Description

(Required) Deinitializes ACL before the application process ends. It is a synchronous interface.
Prototype

```c
aclError aclFinalize()
```

Parameters

None

Returns

0 indicates success. Other values indicate failure.

7.3 Device Management

7.3.1 aclrtSetDevice

Description

Specifies the device used for computation, and implicitly creates the default context, which contains two streams, one default stream, and one stream for internal synchronization. It is a synchronous interface.

If `aclrtSetDevice` is called for multiple times in a process, `aclrtResetDevice` does not need to be called to reset the device resources, because the device resources will be reset when the process exits. It is recommended that `aclrtSetDevice` and `aclrtResetDevice` be used in pairs to reset the device resources that are no longer needed in a timely manner.

The following application scenarios are supported:

- The same device can be specified for computation in different processes or threads.
- If a device is specified in a process, multiple threads in the process can share the device to explicitly create a context (`aclrtCreateContext`).
- In the multi-device scenario, `aclrtSetDevice` can be called in the process to switch to another device. `aclrtSetCurrentContext` is recommended for switching between devices, which is more efficient than using `aclrtSetDevice`.

Prototype

```c
aclError aclrtSetDevice(int32_t deviceId)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deviceId</td>
<td>Input</td>
<td>Device ID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>After <code>aclrtGetDeviceCount</code> is called to obtain the number of available devices, the value range of device ID is [0, number of available devices minus 1].</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.3.2 aclrtResetDevice

Description

Resets the computing device and releases the resources (including the default context and stream, and all streams created in the default context) on the device. It is a synchronous interface. If a task in the default context or stream is not complete, resources will be released after the task is complete.

Restrictions

If there are explicitly created contexts, streams, or events on the device to be reset, you are advised to follow the following calling sequence before resetting. Otherwise, exceptions may occur.

The interfaces calling sequence is as follows: call `aclrtDestroyEvent` to destroy the event or call `aclrtDestroyStream` to destroy the explicitly created stream--> call `aclrtDestroyContext` to destroy the explicitly created context--> call `aclrtResetDevice`.

Prototype

`aclError aclrtResetDevice(int32_t deviceId)`

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deviceId</td>
<td>Input</td>
<td>Device ID</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.
7.3.3 aclrtGetDevice

Description

Obtains the IDs of devices in use. It is a synchronous interface.

Restrictions

If neither aclrtSetDevice nor aclrtCreateContext is called to specify a device, an error is returned when aclrtGetDevice is called.

Prototype

```c
aclError aclrtGetDevice(int32_t *deviceId)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deviceId</td>
<td>Output</td>
<td>Device ID</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.3.4 aclrtGetRunMode

Description

Obtains the current running mode of Ascend AI Software Stack. It is a synchronous interface.

Prototype

```c
aclError aclrtGetRunMode(aclrtRunMode *runMode)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>runMode</td>
<td>Output</td>
<td>Running mode</td>
</tr>
</tbody>
</table>

typedef enum aclrtRunMode {
    //Ascend AI Software Stack running on the Ctrl CPU of the device
    ACL_DEVICE,
    //Ascend AI Software Stack running on the developer board
    ACL_DEVICE,
    //Ascend AI Software Stack running on the host CPU
    ACL_HOST,
} aclrtRunMode;
Returns

0 indicates success. Other values indicate failure.

7.3.5 aclrtSetTsDevice

Description
Sets the Task Schedule required for computation. It is a synchronous interface.

Prototype

```c
aclError aclrtSetTsDevice(aclrtTsId tsId)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tsId</td>
<td>Input</td>
<td>Task Schedule required for computation. This parameter is invalid if only AI core Task Schedule is available in Ascend AI Software Stack. AI core Task Schedule is used by default.</td>
</tr>
</tbody>
</table>

```c
typedef enum aclrtTsId {
    ACL_TS_ID_AICORE = 0, // Use AI core Task Schedule.
    ACL_TS_ID_AIVECTOR = 1, // Use the AIVECTOR Task Schedule.
    ACL_TS_ID_RESERVED = 2,
} aclrtTsId;
```

Returns

0 indicates success. Other values indicate failure.

7.3.6 aclrtGetDeviceCount

Description
Obtains the number of available devices. It is a synchronous interface.

Prototype

```c
aclError aclrtGetDeviceCount(uint32_t *count)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>Output</td>
<td>Number of devices</td>
</tr>
</tbody>
</table>
Returns

0 indicates success. Other values indicate failure.

7.4 Context Management

7.4.1 aclrtCreateContext

Description

Explicitly creates a context, which contains two streams, one default stream, and one stream for internal synchronization. It is a synchronous interface.

The following application scenarios are supported:

- If the aclrtCreateContext interface is not called to display the created context, the system uses the default context, which is created implicitly when aclrtSetDevice is called.
  - Implicitly created context: It is applicable to simple applications without complex interaction logic. However, in multi-thread programming, the execution result depends on the thread scheduling sequence.
  - Explicitly created context (recommended): It is applicable to large-scale applications with complex interaction logic, improving application readability and maintainability.
- If multiple contexts are created in a process, the current thread can use only one context at a time. It is recommended that aclrtGetCurrentContext be used to specify the context of the current thread to improve program maintainability. (The number of contexts depends on the number of streams. For details, see 7.5.1 aclrtCreateStream.)

**NOTE**

If the aclrtSetDevice API is not called by the application, when the aclrtCreateContext API is called for the first time, the system binds a default stream to the device based on the device ID passed by the API (one device is bound to one default stream). Therefore, only when the aclrtCreateContext API is called for the first time, three streams are occupied: the default stream bound to the device, the default stream in the context, and the stream used for internal synchronization in the context.

Prototype

```c
aclError aclrtCreateContext(aclrtContext *context, int32_t deviceId)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deviceId</td>
<td>Input</td>
<td>Device ID in which a context is to be created</td>
</tr>
<tr>
<td>context</td>
<td>Output</td>
<td>Pointer to the created context</td>
</tr>
</tbody>
</table>
Returns

0 indicates success. Other values indicate failure.

7.4.2 aclrtDestroyContext

Description

Destroys a context and releases the resources of the context. It is a synchronous interface. Only the context created by calling aclrtCreateContext can be destroyed.

Prototype

aclError aclrtDestroyContext(aclrtContext context)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>Input</td>
<td>Context to be destroyed</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.4.3 aclrtSetCurrentContext

Description

Sets context of a thread. It is a synchronous interface.

The following application scenarios are supported:

- If you explicitly create a context (for example, ctx1) by calling aclrtCreateContext in a thread (for example, thread1), you do not need to call the aclrtSetCurrentContext interface to specify the context of the thread. The system uses ctx1 as the context of thread1 by default.
- If aclrtCreateContext is not called to display the created context, the system uses the default context as the context of the thread. In this case, the default context cannot be released by calling aclrtDestroyContext.
- If aclrtSetCurrentContext is called for multiple times to set the context of the thread, the last setting prevails.

Restrictions

- If the device corresponding to the context set for the thread has been reset, the context cannot be set as the thread context. Otherwise, exceptions may occur.
- It is recommended that the context created in a thread be used in the thread. If thread A calls aclrtCreateContext to create a context and thread B uses
this context, you need to ensure the tasks execution sequence of the two threads that are in the same stream of the same context.

Prototype

```c
aclError aclrtSetCurrentContext(aclrtContext context)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>Input</td>
<td>Current context of the thread</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.4.4 aclrtGetCurrentContext

Description

Obtains the context of a thread. It is a synchronous interface.

If you call `aclrtSetCurrentContext` multiple times to set the context of the current thread, the context set last time is obtained.

Prototype

```c
aclError aclrtGetCurrentContext(aclrtContext *context)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>Output</td>
<td>Current context of the thread</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.5 Streams Management

7.5.1 aclrtCreateStream

Description

Creates a stream. It is a synchronous interface.
Restrictions

The hardware supports a maximum of 1024 streams. If there are multiple default streams, only explicit streams can be created. The number of allowed explicit streams is calculated as follows: 1024 – Number of default streams – Number of streams for internal synchronization. For example, if a default stream and a stream for internal synchronization already exist, only 1022 (= 1024 – 1 – 1) streams can be created explicitly.

Each context corresponds to a default stream, which is implicitly created by calling aclrtSetDevice or aclrtCreateContext. You are advised to explicitly create a stream by calling aclrtCreateStream.

- Implicitly created stream: It is applicable to simple applications without complex interaction logic. However, in multi-thread programming, the execution result depends on the thread scheduling sequence.
- Explicitly created stream (recommended): It is applicable to large-scale applications with complex interaction logic, improving application readability and maintainability.

Prototype

```
aclError aclrtCreateStream(aclrtStream *stream)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stream</td>
<td>Output</td>
<td>Pointer to the created stream</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.5.2 aclrtDestroyStream

Description

Destroys a specified stream. Only streams created by calling aclrtCreateStream can be destroyed. It is a synchronous interface.

Restrictions

- Before calling aclrtDestroyStream to destroy a specified stream, call aclrtSynchronizeStream to ensure that all tasks in the stream are completely executed.
- When calling aclrtDestroyStream to destroy a specified stream, ensure that the stream is in the current context.

Prototype

```
aclError aclrtDestroyStream(aclrtStream stream)
```
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stream</td>
<td>Input</td>
<td>Stream to be destroyed</td>
</tr>
</tbody>
</table>

### Returns

0 indicates success. Other values indicate failure.

### 7.6 Synchronous Wait

#### 7.6.1 aclrtCreateEvent

**Description**

Creates an event. It is a synchronous interface.

**Prototype**

```c
aclError aclrtCreateEvent(aclrtEvent *event)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>Output</td>
<td>Pointer to the created event</td>
</tr>
</tbody>
</table>

### Returns

0 indicates success. Other values indicate failure.

#### 7.6.2 aclrtDestroyEvent

**Description**

Destroys an event. Only the events created by calling `aclrtCreateEvent` can be destroyed. It is a synchronous interface. Before destroying an event, ensure that the tasks involved in `aclrtSynchronizeEvent` or `aclrtStreamWaitEvent` are complete.

**Prototype**

```c
aclError aclrtDestroyEvent(aclrtEvent event)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>Input</td>
<td>Event to be destroyed</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.6.3 aclrtRecordEvent

Description

Records an event in a stream. It is a synchronous interface.

`aclrtRecordEvent` and `aclrtStreamWaitEvent` are used in conjunction for synchronization between streams. When `aclrtRecordEvent` is called, an event resource is allocated. After `aclrtStreamWaitEvent` is called, `aclrtResetEvent` should be called to reset the event resource in a timely manner.

API calling sequence:

```
aclrtCreateEvent -> aclrtRecordEvent -> aclrtStreamWaitEvent -> aclrtResetEvent
```

Prototype

```c
aclError aclrtRecordEvent(aclrtEvent event, aclrtStream stream)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>Input</td>
<td>Event to be recorded</td>
</tr>
<tr>
<td>stream</td>
<td>Input</td>
<td>Stream where the event will be recorded in Pass <strong>NULL</strong> if the default stream is used.</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.6.4 aclrtResetEvent

Description

Resets an event. Resets the event only after all tasks in the stream are completely executed. It is a synchronous interface.
Restrictions

Resetting an event also resets the event status. The event status is recorded when the aclrtRecordEvent API is called. Therefore, before calling aclrtResetEvent, you need to call the aclrtRecordEvent API first.

The API calling sequence is as follows: aclrtCreateEvent -> aclrtRecordEvent -> aclrtResetEvent

Prototype

aclError aclrtResetEvent(aclrtEvent event, aclrtStream stream)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>Input</td>
<td>Event to be reset</td>
</tr>
<tr>
<td>stream</td>
<td>Input</td>
<td>Stream where the event is in</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.6.5 aclrtQueryEvent

Description

Queries the status of an event. It is a synchronous interface.

Prototype

aclError aclrtQueryEvent(aclrtEvent event, aclrtEventStatus *status)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>Input</td>
<td>Event to be queried</td>
</tr>
<tr>
<td>status</td>
<td>Output</td>
<td>Event status</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.
7.6.6 aclrtSynchronizeEvent

Description
Blocks application execution and waits for the event to be complete. It is a synchronous interface.

Prototype

```plaintext
aclError aclrtSynchronizeEvent(aclrtEvent event)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>Input</td>
<td>Event to be waited</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.6.7 aclrtEventElapsedTime

Description
Collects statistics on the elapsed time between two events. It is a synchronous interface.

Restrictions
The API calling sequence is as follows: call aclrtCreateEvent to create an event-> call aclrtRecordEvent to record the start event and end event in the same stream-> call aclrtSynchronizeStream to block application execution until all tasks in the specified stream are complete-> call aclrtEventElapsedTime to collect statistics on the elapsed time between two events.

Prototype

```plaintext
aclError aclrtEventElapsedTime(float *ms, aclrtEvent start, aclrtEvent end)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ms</td>
<td>Output</td>
<td>Elapsed time between the two events, in milliseconds</td>
</tr>
<tr>
<td>start</td>
<td>Input</td>
<td>Start event</td>
</tr>
<tr>
<td>end</td>
<td>Input</td>
<td>End event</td>
</tr>
</tbody>
</table>
Returns

0 indicates success. Other values indicate failure.

7.6.8 aclrtStreamWaitEvent

Description

Blocks the execution of a stream until the specified event is complete. It is a synchronous interface. Supports scenarios where multiple streams wait for the same event.

`aclrtRecordEvent` and `aclrtStreamWaitEvent` are used in conjunction for synchronization between streams. When `aclrtRecordEvent` is called, an event resource is allocated. After `aclrtStreamWaitEvent` is called, `aclrtResetEvent` should be called to reset the event resource in a timely manner.

API calling sequence: `aclrtCreateEvent` -> `aclrtRecordEvent` -> `aclrtStreamWaitEvent` -> `aclrtResetEvent`

Prototype

```c
aclError aclrtStreamWaitEvent(aclrtStream stream, aclrtEvent event)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stream</td>
<td>Input</td>
<td>Stream that needs to wait for the event to complete Pass <strong>NULL</strong> if the default stream is used.</td>
</tr>
<tr>
<td>event</td>
<td>Input</td>
<td>Event to be waited</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.6.9 aclrtSynchronizeDevice

Description

Blocks application execution until the ongoing computation on the device is complete. It is a synchronous interface.

In the multi-device scenario, the corresponding device of the current context is to be waited.

Prototype

```c
aclError aclrtSynchronizeDevice(void)
```
Parameters

None

Returns

0 indicates success. Other values indicate failure.

7.6.10 aclrtSynchronizeStream

Description

Block application execution until all tasks in a specified stream are complete. It is a synchronous interface.

Prototype

\[
\text{aclError aclrtSynchronizeStream(aclrtStream stream)}
\]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stream</td>
<td>Input</td>
<td>Stream that needs to complete all tasks</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.6.11 aclrtSubscribeReport

Description

Specifies the thread that processes the callback function in the stream. It is a synchronous interface.

Prototype

\[
\text{aclError aclrtSubscribeReport(uint64_t threadId, aclrtStream stream)}
\]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>threadId</td>
<td>Input</td>
<td>Thread ID</td>
</tr>
<tr>
<td>stream</td>
<td>Input</td>
<td>Stream</td>
</tr>
</tbody>
</table>
Returns

0 indicates success. Other values indicate failure.

7.6.12 aclrtLaunchCallback

Description

Adds a callback function executed on the host or device to the stream task queue. It is an asynchronous interface.

Prototype

```c
aclError aclrtLaunchCallback(aclrtCallback fn, void *userData, aclrtCallbackBlockType blockType, aclrtStream stream)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fn</td>
<td>Input</td>
<td>Callback function to be added</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The prototype of the callback function is as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>typedef void (*aclrtCallback)(void *userData)</td>
</tr>
<tr>
<td>userData</td>
<td>Input</td>
<td>User data to be passed to the callback function</td>
</tr>
<tr>
<td>blockType</td>
<td>Input</td>
<td>Whether the specified callback function blocks device execution</td>
</tr>
<tr>
<td></td>
<td></td>
<td>typedef enum aclrtCallbackBlockType {</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACL_CALLBACK_NO_BLOCK, // Not blocked</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACL_CALLBACK_BLOCK, // Blocked</td>
</tr>
<tr>
<td></td>
<td></td>
<td>} aclrtCallbackBlockType;</td>
</tr>
<tr>
<td>stream</td>
<td>Input</td>
<td>Stream</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.6.13 aclrtProcessReport

Description

Triggers callback handling after a specified period (thread specified by `aclrtSubscribeReport` handles the callback). It is a synchronous interface.

A thread needs to be created to call `aclrtProcessReport` in the thread function.
Prototype

```c
aclError aclrtProcessReport(int32_t timeout)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout</td>
<td>Input</td>
<td>Timeout period, in milliseconds</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value range:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>-1</code>: infinite waiting</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Values greater than 0 (excluding 0): the waiting period</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.6.14 aclrtUnSubscribeReport

Description

Unsubscribe a thread and specifies a callback function in a stream to not be handled by a specified thread. It is a synchronous interface.

Prototype

```c
aclError aclrtUnSubscribeReport(uint64_t threadId, aclrtStream stream)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>threadId</td>
<td>Input</td>
<td>Thread ID</td>
</tr>
<tr>
<td>stream</td>
<td>Input</td>
<td>Stream</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.6.15 aclrtSetExceptionInfoCallback

Description

Sets the exception callback function. It is a synchronous interface.

- Before executing a task, you need to set an exception callback function. When an exception occurs during task execution on the device, the system passes a
**aclrtExceptionInfo** struct pointer that contains the task ID, stream ID, and thread ID to the exception callback function and executes the callback function. You can call **aclrtGetTaskIdFromExceptionInfo**, **aclrtGetStreamIdFromExceptionInfo**, and **aclrtGetThreadIdFromExceptionInfo** to obtain the error task ID, stream ID, and thread ID, facilitating fault locating.

- If the exception callback function is set for multiple times, the last setting applies.
- To clear the callback function, pass a null pointer to the **aclrtSetExceptionInfoCallback** call.

Example use case: Before calling **aclopExecute**, call **aclrtSetExceptionInfoCallback** to set an exception callback function. When operator execution on the device fails, the system passes a **aclrtExceptionInfo** struct pointer that contains the task ID, stream ID, and thread ID to the exception callback function and executes the callback function.

**Prototype**

```c
aclError aclrtSetExceptionInfoCallback(aclrtExceptionInfoCallback callback)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>callback</td>
<td>Input</td>
<td>Callback function</td>
</tr>
<tr>
<td>Prototype:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>void (*aclrtExceptionInfoCallback)(aclrtExceptionInfo *exceptionInfo);</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

0 indicates success. Other values indicate failure.

### 7.6.16 aclrtGetTaskIdFromExceptionInfo

**Description**

Obtains the task ID in the exception information. It is a synchronous interface.

**Prototype**

```c
uint32_t aclrtGetTaskIdFromExceptionInfo(const aclrtExceptionInfo *info)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>info</td>
<td>Input</td>
<td>Exception information</td>
</tr>
</tbody>
</table>

If `aclrtSetExceptionInfoCallback` is called before an asynchronous task is executed, the system stores the task ID, stream ID, and thread ID of the exception in the `aclExceptionInfo` struct.

Returns

The task ID in the returned exception information. `0xFFFFFFFF` (hexadecimal) indicates that the exception information is empty.

7.6.17 aclrtGetStreamIdFromExceptionInfo

Description

Obtains the stream ID in the exception information. It is a synchronous interface.

Prototype

```
uint32_t aclrtGetStreamIdFromExceptionInfo(const aclrtExceptionInfo *info)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>info</td>
<td>Input</td>
<td>Exception information</td>
</tr>
</tbody>
</table>

If `aclrtSetExceptionInfoCallback` is called before an asynchronous task is executed, the system stores the task ID, stream ID, and thread ID of the exception in the `aclExceptionInfo` struct.

Returns

The stream ID in the returned exception information. `0xFFFFFFFF` (hexadecimal) indicates that the exception information is empty.

7.6.18 aclrtGetThreadIdFromExceptionInfo

Description

Obtains the thread ID in the exception information. It is a synchronous interface.
Prototype

```c
uint32_t aclrtGetThreadIdFromExceptionInfo(const aclrtExceptionInfo *info)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>info</td>
<td>Input</td>
<td>Exception information. If <code>aclrtSetExceptionInfoCallback</code> is called before an asynchronous task is executed, the system stores the task ID, stream ID, and thread ID of the exception in the <code>aclExceptionInfo</code> struct.</td>
</tr>
</tbody>
</table>

Returns

The thread ID in the returned exception information. `0xFFFFFFFF` (hexadecimal) indicates that the exception information is empty.

7.7 Memory Management

7.7.1 aclrtMalloc

Description

Allocates memory on the device. It is a synchronous interface.

Allocates linear memory of the specified `size` on the device and returns the pointer to allocated memory (*devPtr).

Before calling a media data processing API, `acldvppMalloc` must be called to allocate memory on the device for storing the input or output data.

Restrictions

- If the memory is allocated by using the `aclrtMalloc` interface, the memory needs to be released by calling `aclrtFree`.
- Performance deterioration will be caused by the frequent calling of `aclrtMalloc` to allocate memory and `aclrtFree` to free memory. You are advised to allocate or manage memory in advance to avoid frequent memory application and freeing.
- The allocated memory size is the input `size` rounded up to the nearest multiple of 32, plus 32 bytes.

Prototype

```c
aclError aclrtMalloc(void **devPtr, size_t size, aclrtMemMallocPolicy policy)
```
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>Input</td>
<td>Allocated memory size, in bytes Must not be 0.</td>
</tr>
<tr>
<td>policy</td>
<td>Input</td>
<td>Memory allocation policy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>typedef enum aclrtMemMallocPolicy {</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACL_MEM_MALLOC_HUGE_FIRST, // Prioritize the allocation of huge page memory. If huge page memory is insufficient, allocate normal page memory.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACL_MEM_MALLOC_HUGE_ONLY, // Allocate only huge page memory. If huge page memory is insufficient, an error is returned.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACL_MEM_MALLOC_NORMAL_ONLY, // Allocate only normal page memory.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>} aclrtMemMallocPolicy;</td>
</tr>
<tr>
<td>devPtr</td>
<td>Output</td>
<td>Pointer to the pointer to the allocated memory on the device</td>
</tr>
</tbody>
</table>

### Returns

**0** indicates success. Other values indicate failure.

### 7.7.2 aclrtFree

#### Description

Frees the memory on the device. It is a synchronous interface.

`aclrtFree` can free only the memory allocated by using `aclrtMalloc`.

#### Prototype

`aclError aclrtFree(void *devPtr)`

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>devPtr</td>
<td>Input</td>
<td>Pointer to the memory to be freed</td>
</tr>
</tbody>
</table>

### Returns

**0** indicates success. Other values indicate failure.
7.7.3 aclrtMallocHost

Description

Allocates memory on the host. It is a synchronous interface.

Restrictions

- The allocated memory cannot be used on the device and needs to be explicitly copied to the device.
- If the memory is allocated by using the aclrtMallocHost interface, the memory needs to be released by calling aclrtFreeHost.

Prototype

```
aclError aclrtMallocHost(void **hostPtr, size_t size)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>Input</td>
<td>Allocated memory size, in bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Must not be 0.</td>
</tr>
<tr>
<td>hostPtr</td>
<td>Output</td>
<td>Pointer to the allocated memory</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.7.4 aclrtFreeHost

Description

Frees the memory on the host. It is a synchronous interface.

aclrtFreeHost can free only the memory allocated by using aclrtMallocHost.

Prototype

```
aclError aclrtFreeHost(void *hostPtr)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostPtr</td>
<td>Input</td>
<td>Pointer to the memory to be freed</td>
</tr>
</tbody>
</table>
Returns

0 indicates success. Other values indicate failure.

7.7.5 aclrtMemset

Description

Initializes the memory and sets the content in the memory to a specified value. It is a synchronous interface.

The system determines whether the memory to be initialized is on the host or device side based on the address.

Prototype

```c
aclError aclrtMemset (void *devPtr, size_t maxCount, int32_t value, size_t count)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>devPtr</td>
<td>Input</td>
<td>Start address of the memory</td>
</tr>
<tr>
<td>maxCount</td>
<td>Input</td>
<td>Maximum memory size, in bytes</td>
</tr>
<tr>
<td>value</td>
<td>Input</td>
<td>Value to be set</td>
</tr>
<tr>
<td>count</td>
<td>Input</td>
<td>Memory size to be set, in bytes</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.7.6 aclrtMemsetAsync

Description

Initializes the memory and sets the content in the memory to a specified value.

It is an asynchronous interface. The successful calling of this interface indicates only the success of the task delivery, not the task execution. After calling this interface, `aclrtSynchronizeStream` must be called to ensure that the memory initialization task is complete.

The system determines whether the memory to be initialized is on the host or device side based on the address.

Prototype

```c
aclError aclrtMemsetAsync(void *devPtr, size_t maxCount, int32_t value, size_t count, aclrtStream stream)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>devPtr</td>
<td>Input</td>
<td>Start address of the memory</td>
</tr>
<tr>
<td>maxCount</td>
<td>Input</td>
<td>Maximum memory size, in bytes</td>
</tr>
<tr>
<td>value</td>
<td>Input</td>
<td>Value to be set</td>
</tr>
<tr>
<td>count</td>
<td>Input</td>
<td>Memory size to be set, in bytes</td>
</tr>
<tr>
<td>stream</td>
<td>Input</td>
<td>Stream ID</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.7.7 aclrtMemcpy

Description

Implements synchronous memory copy between the host and device sides.

The system determines whether to copy data from the source address to the destination address based on the pointer to the source and destination memory address. If not, an error is reported.

Prototype

```c
aclError aclrtMemcpy(void *dst, size_t destMax, const void *src, size_t count, aclrtMemcpyKind kind)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dst</td>
<td>Input</td>
<td>Pointer to the destination memory address</td>
</tr>
<tr>
<td>destMax</td>
<td>Input</td>
<td>Maximum size of the destination memory address, in bytes</td>
</tr>
<tr>
<td>src</td>
<td>Input</td>
<td>Pointer to the source memory address</td>
</tr>
<tr>
<td>count</td>
<td>Input</td>
<td>Size of the copied memory, in bytes</td>
</tr>
</tbody>
</table>
### Returns

0 indicates success. Other values indicate failure.

#### 7.7.8 aclrtMemcpyAsync

**Description**

Implements asynchronous memory copy between the host and device.

It is an asynchronous interface. The successful calling of this interface indicates only the success of the task delivery, not the task execution. After calling this interface, **aclrtSynchronizeStream** must be called to ensure that the memory copy task is complete.

**Restrictions**

For on-chip memory copy within the device, the source and destination address must be 64-byte aligned.

**Prototype**

```c
aclError aclrtMemcpyAsync(void *dst, size_t destMax, const void *src, size_t count, aclrtMemcpyKind kind, aclrtStream stream)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dst</td>
<td>Input</td>
<td>Pointer to the destination memory address</td>
</tr>
<tr>
<td>destMax</td>
<td>Input</td>
<td>Maximum size of the destination memory address, in bytes</td>
</tr>
<tr>
<td>src</td>
<td>Input</td>
<td>Pointer to the source memory address</td>
</tr>
<tr>
<td>Parameter</td>
<td>Input/Output</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>count</td>
<td>Input</td>
<td>Size of the copied memory, in bytes</td>
</tr>
<tr>
<td>kind</td>
<td>Input</td>
<td>Memory copy kind</td>
</tr>
<tr>
<td></td>
<td></td>
<td>typedef enum aclrtMemcopyKind {</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACL_MEMCPY_HOST_TO_HOST, // Memory copy within the host side</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACL_MEMCPY_HOST_TO_DEVICE, // Host to device memory copy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACL_MEMCPY_DEVICE_TO_HOST, // Device to host memory copy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Memory replication in the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACL_MEMCPY_DEVICE_TO_DEVICE, // Memory copy within the device side</td>
</tr>
<tr>
<td></td>
<td></td>
<td>} aclrtMemcopyKind;</td>
</tr>
<tr>
<td>stream</td>
<td>Input</td>
<td>Stream ID</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.8 Model Loading and Execution

7.8.1 aclmdlLoadFromFile

Description

Loads offline model data from a file (offline model adapted to the Ascend AI Processor). The memory is managed by the system. It is a synchronous interface.

Returns model ID after the model is loaded. The model ID is used for model identification in the follow-up operations.

Prototype

`aclError aclmdlLoadFromFile(const char *modelPath, uint32_t *modelId)`

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelPath</td>
<td>Input</td>
<td>Path for storing the offline model file. The file name is contained in the path. The user who runs the application must have the permission to access the path. The offline model file is an offline model adapted to the Ascend AI Processor, that is, an *.om file.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Input/Output</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>modelId</td>
<td>Output</td>
<td>Model ID generated after the model is loaded</td>
</tr>
</tbody>
</table>

**Returns**

0 indicates success. Other values indicate failure.

### 7.8.2 aclmdlLoadFromMem

**Description**

Loads offline model data from memory. The model execution memory is managed by the system. It is a synchronous interface.

Returns model ID after the model is loaded. The model ID is used for model identification in the follow-up operations.

**Prototype**

```c
aclError aclmdlLoadFromMem(const void* model, size_t modelSize, uint32_t* modelId)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>Input</td>
<td>Memory address of the model data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Allocate memory on the host or device based on the application runs. Obtain the application running mode by calling aclrtGetRunMode.</td>
</tr>
<tr>
<td>modelSize</td>
<td>Input</td>
<td>Size of the model data in the memory, in bytes</td>
</tr>
<tr>
<td>modelId</td>
<td>Output</td>
<td>Model ID generated after the model is loaded</td>
</tr>
</tbody>
</table>

**Returns**

0 indicates success. Other values indicate failure.
7.8.3 aclmdlLoadFromFileWithMem

Description

Loads offline model data from a file (offline model adapted to the Ascend AI Processor). The model execution memory is managed by the user. It is a synchronous interface.

Returns model ID after the model is loaded. The model ID is used for model identification in the follow-up operations.

Prototype

```c
aclError aclmdlLoadFromFileWithMem(const char *modelPath, uint32_t *modelId, void *workPtr, size_t workSize, void *weightPtr, size_t weightSize)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelPath</td>
<td>Input</td>
<td>Path for storing the offline model file. The file name is contained in the path. The user who runs the application must have the permission to access the storage path. The offline model file is an offline model adapted to the Ascend AI Processor, that is, an *.om file.</td>
</tr>
<tr>
<td>modelId</td>
<td>Output</td>
<td>Model ID generated after the model is loaded</td>
</tr>
<tr>
<td>workPtr</td>
<td>Input</td>
<td>Address of the working memory (for storing model input and output data) required by the model on the device. The address is managed by the user.</td>
</tr>
<tr>
<td>workSize</td>
<td>Input</td>
<td>Working memory size required by the model, in bytes</td>
</tr>
<tr>
<td>weightPtr</td>
<td>Input</td>
<td>Pointer to the model weight memory (for storing weight data) on the device. The pointer is managed by the user.</td>
</tr>
<tr>
<td>weightSize</td>
<td>Input</td>
<td>Weight memory size required by the model, in bytes</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.
7.8.4 aclmdlLoadFromMemWithMem

Description

Loads offline model data from memory. The model execution memory is managed by the user. It is a synchronous interface.

Returns model ID after the model is loaded. The model ID is used for model identification in the follow-up operations.

Prototype

```c
aclError aclmdlLoadFromMemWithMem(const void *model, size_t modelSize, uint32_t *modelId, void *workPtr, size_t workSize, void *weightPtr, size_t weightSize)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
</table>
| model         | Input        | Memory address of the model data. Allocate memory on the host or device based on where the application runs. Obtain the application running mode by calling `aclrtGetRunMode`.
| modelSize     | Input        | Model data size, in bytes |
| modelId       | Output       | Model ID generated after the model is loaded |
| workPtr       | Input        | Address of the working memory (for storing model input and output data) required by the model on the device. The address is managed by the user, and it can be empty. |
| workSize      | Input        | Working memory size required by the model, in bytes. This parameter is invalid if `workPtr` is empty. |
| weightPtr     | Input        | Pointer to the model weight memory (for storing weight data) on the device. The pointer is managed by the user, and it can be empty. |
| weightSize    | Input        | Weight memory size required by the model, in bytes. This parameter is invalid if `weightPtr` is empty. |
Returns

0 indicates success. Other values indicate failure.

7.8.5 aclmdlLoadFromFileWithQ

Description

Loads offline model data from a file (offline model adapted to the Ascend AI Processor). The input and output model data is stored in the queue. It is a synchronous interface. The current version does not support this interface.

Returns model ID after the model is loaded. The model ID is used for model identification in the follow-up operations.

Prototype

```c
aclError aclmdlLoadFromFileWithQ(const char *modelPath, uint32_t *modelId, const uint32_t *inputQ, size_t inputQNum, const uint32_t *outputQ, size_t outputQNum)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelPath</td>
<td>Input</td>
<td>Path for storing the offline model file. The file name is contained in the path. The user who runs the application must have the permission to access the storage path. The offline model file is an offline model adapted to the Ascend AI Processor, that is, an *.om file.</td>
</tr>
<tr>
<td>modelId</td>
<td>Output</td>
<td>Model ID generated after the model is loaded</td>
</tr>
<tr>
<td>inputQ</td>
<td>Input</td>
<td>Queue ID. Each input model corresponds to a queue ID.</td>
</tr>
<tr>
<td>inputQNum</td>
<td>Input</td>
<td>Number of input queues</td>
</tr>
<tr>
<td>outputQ</td>
<td>Input</td>
<td>Queue ID. Each output model corresponds to a queue ID.</td>
</tr>
<tr>
<td>outputQNum</td>
<td>Input</td>
<td>Number of output queues</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.
### 7.8.6 aclmdlLoadFromMemWithQ

**Description**

Loads offline model data from memory (offline model adapted to the Ascend AI Processor). The input and output model data is stored in the queue. It is a synchronous interface. The current version does not support this interface.

Returns model ID after the model is loaded. The model ID is used for model identification in the follow-up operations.

**Prototype**

```c
aclError aclmdlLoadFromMemWithQ(const void *model, size_t modelSize, uint32_t *modelId,
                                const uint32_t *inputQ, size_t inputQNum, const uint32_t *outputQ, size_t outputQNum)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>Input</td>
<td>Memory address of the model data</td>
</tr>
<tr>
<td>modelSize</td>
<td>Input</td>
<td>Size of the model data in the memory, in bytes</td>
</tr>
<tr>
<td>modelId</td>
<td>Output</td>
<td>Model ID generated after the model is loaded</td>
</tr>
<tr>
<td>inputQ</td>
<td>Input</td>
<td>Queue ID. Each input model corresponds to a queue ID.</td>
</tr>
<tr>
<td>inputQNum</td>
<td>Input</td>
<td>Number of input queues</td>
</tr>
<tr>
<td>outputQ</td>
<td>Input</td>
<td>Queue ID. Each output model corresponds to a queue ID.</td>
</tr>
<tr>
<td>outputQNum</td>
<td>Input</td>
<td>Number of output queues</td>
</tr>
</tbody>
</table>

**Returns**

0 indicates success. Other values indicate failure.

### 7.8.7 aclmdlExecute

**Description**

Executes model inference until the result is returned. It is a synchronous interface.
Prototype

```c
aclError aclmdlExecute(uint32_t modelId, const aclmdlDataset *input,
aclmdlDataset *output)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelId</td>
<td>Input</td>
<td>Model ID for inference. A successful <code>aclmdlLoadFromFile</code>, <code>aclmdlLoadFromMem</code>, <code>aclmdlLoadFromFileWithMem</code>, or <code>aclmdlLoadFromMemWithMem</code> call returns a model ID.</td>
</tr>
<tr>
<td>input</td>
<td>Input</td>
<td>Input data for model inference. If the huge memory allocated by calling <code>aclrtMalloc</code> is divided and managed by the user, the user needs to align and round up the input data. Specifically, the start address must be 128-byte aligned, and the input data plus 32 bytes must be rounded up to the nearest multiple of 128 bytes.</td>
</tr>
<tr>
<td>output</td>
<td>Output</td>
<td>Output data of model inference. If the huge memory allocated by calling <code>aclrtMalloc</code> is divided and managed by the user, the user needs to align and round up the output data. Specifically, the start address must be 128-byte aligned, and the output data plus 32 bytes must be rounded up to the nearest multiple of 128 bytes.</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

**7.8.8 aclmdlExecuteAsync**

Description

Executes model inference. It is an asynchronous interface.
Prototype

```c
aclError aclmdlExecuteAsync(uint32_t modelId, const aclmdlDataset *input, aclmdlDataset *output, aclrtStream stream)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
</table>
| modelId   | Input        | Model ID for inference  
A successful `aclmdlLoadFromFile`, `aclmdlLoadFromMem`, `aclmdlLoadFromFileWithMem`, or `aclmdlLoadFromMemWithMem` call returns a model ID. |
| input     | Input        | Input data for model inference  
If the huge memory allocated by calling `aclrtMalloc` is divided and managed by the user, the user needs to align and round up the input data. Specifically, the start address must be 128-byte aligned, and the input data plus 32 bytes must be rounded up to the nearest multiple of 128 bytes. |
| output    | Output       | Output data of model inference  
If the huge memory allocated by calling `aclrtMalloc` is divided and managed by the user, the user needs to align and round up the output data. Specifically, the start address must be 128-byte aligned, and the output data plus 32 bytes must be rounded up to the nearest multiple of 128 bytes. |
| stream    | Input        | Stream |

Returns

0 indicates success. Other values indicate failure.

7.8.9 aclmdlUnload

Description

Uninstalls the model and releases resources after the model inference is complete. It is a synchronous interface.
Prototype

```c
aclError aclmdlUnload(uint32_t modelId)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelId</td>
<td>Input</td>
<td>Model ID to be uninstalled</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

### 7.8.10 aclmdlQuerySize

**Description**

Obtains the weight memory size and working memory size required for model execution based on the model file. It is a synchronous interface.

Prototype

```c
aclError aclmdlQuerySize(const char *fileName, size_t *workSize, size_t *weightSize)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileName</td>
<td>Input</td>
<td>Path of the model that needs to obtain memory information. The file name is contained in the path. The user who runs the application must have the permission to access the path.</td>
</tr>
<tr>
<td>workSize</td>
<td>Output</td>
<td>Size of the working memory required for model execution, in bytes</td>
</tr>
<tr>
<td>weightSize</td>
<td>Output</td>
<td>Size of the weight memory required for model execution, in bytes</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.
### 7.8.11 aclmdlQuerySizeFromMem

**Description**

Obtains the weight memory size and memory size required for model execution based on the model data in the memory. It is a synchronous interface.

**Restrictions**

The execution and weight memory is on the device, which needs to be allocated and released by the user.

**Prototype**

```c
aclError aclmdlQuerySizeFromMem(const void *model, size_t modelSize, size_t *workSize, size_t *weightSize)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>Input</td>
<td>Model data that needs to obtain memory information</td>
</tr>
<tr>
<td>modelSize</td>
<td>Input</td>
<td>Model data size, in bytes</td>
</tr>
<tr>
<td>workSize</td>
<td>Output</td>
<td>Working memory size required for model execution, in bytes</td>
</tr>
<tr>
<td>weightSize</td>
<td>Output</td>
<td>Weight memory size required for model execution, in bytes</td>
</tr>
</tbody>
</table>

**Returns**

0 indicates success. Other values indicate failure.

### 7.8.12 aclmdlSetDynamicBatchSize

**Description**

Sets the dynamic batch size choices for model inference in dynamic batch size scenarios.

The batch size comes from the batch size choices specified by the `dynamic_batch_size` parameter during model conversion. For details about model conversion, see ATC Tool Instructions.

You can also call the `aclmdlGetDynamicBatch` API to obtain the number of batch size choices as well as the batch size of each choice supported by the model.
Prototype

```c
aclError aclmdlSetDynamicBatchSize(uint32_t modelId, aclmdlDataset *dataset, size_t index, uint64_t batchSize)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelId</td>
<td>Input</td>
<td>Model ID. A successful <code>aclmdlLoadFromFile</code>, <code>aclmdlLoadFromMem</code>, <code>aclmdlLoadFromFileWithMem</code>, or <code>aclmdlLoadFromMemWithMem</code> call returns a model ID.</td>
</tr>
<tr>
<td>dataset</td>
<td>Input</td>
<td>Input data of a model. Data of type <code>aclmdlDataset</code> is used to describe input data for model inference, while data of type <code>aclDataBuffer</code> is used to describe input memory size and address.</td>
</tr>
<tr>
<td>index</td>
<td>Input</td>
<td>Index of the input batch size. Index is obtained by calling the <code>aclmdlGetInputIndexByName</code> interface. The input name is fixed to <code>ACL_DYNAMIC_TENSOR_NAME</code>.</td>
</tr>
<tr>
<td>batchSize</td>
<td>Input</td>
<td>Batch size for model inference.</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.8.13 aclmdlSetDynamicHWSize

Description

Sets the dynamic image size choices for model inference in dynamic image size scenarios.

The image size comes from the image size choices specified by the `dynamic_image_size` parameter during model conversion. For details about model conversion, see ATC Tool Instructions.

You can also call the `aclmdlGetDynamicHW` API to obtain the number of image size choices as well as the image size of each choice supported by the model.

Prototype

```c
aclError aclmdlSetDynamicHWSize(uint32_t modelId, aclmdlDataset *dataset, size_t index, uint64_t height, uint64_t width)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelId</td>
<td>Input</td>
<td>Model ID A successful aclmdlLoadFromFile, aclmdlLoadFromMem, aclmdlLoadFromFileWithMem, or aclmdlLoadFromMemWithMem call returns a model ID.</td>
</tr>
<tr>
<td>dataset</td>
<td>Input</td>
<td>Input data of a model Data of type aclmdlDataset is used to describe input data for model inference, while data of type aclDataBuffer is used to describe input memory size and address.</td>
</tr>
<tr>
<td>index</td>
<td>Input</td>
<td>Index of the input image size. Index is obtained by calling the aclmdlGetInputIndexByName interface. The input name is fixed to ACL_DYNAMIC_TENSOR_NAME.</td>
</tr>
<tr>
<td>height</td>
<td>Input</td>
<td>Height to be set</td>
</tr>
<tr>
<td>width</td>
<td>Input</td>
<td>Width to be set</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.8.14 aclmdlSetInputAIPP

Description

Sets AIPP parameters for model inference in dynamic AIPP scenarios. It is a synchronous interface.

Prototype

aclError aclmdlSetInputAIPP(uint32_t modelId, aclmdlDataset *dataset, size_t index, const aclmdlAIPP *aippParmsSet)
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelId</td>
<td>Input</td>
<td>Model ID A successful aclmdlLoadFromFile, aclmdlLoadFromMem, aclmdlLoadFromFileWithMem, or aclmdlLoadFromMemWithMem call returns a model ID.</td>
</tr>
<tr>
<td>dataset</td>
<td>Input</td>
<td>Input data of a model Data of type aclmdDataset is used to describe input data for model inference, while data of type aclDataBuffer is used to describe input memory size and address.</td>
</tr>
<tr>
<td>index</td>
<td>Input</td>
<td>Index of the input AIPP. Index is obtained by calling the aclmdlGetInputIndexByName interface. The input name is fixed to ACL_DYNAMIC_AIPP_NAME.</td>
</tr>
<tr>
<td>aippParmsSet</td>
<td>Input</td>
<td>Dynamic AIPP parameter object Call aclmdlCreateAIPP in advance to create data of the aclmdIAIPP type.</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.9 Operator Building

7.9.1 aclopRegisterCompileFunc

Description

Registers the operator selector in dynamic-shape scenarios, so that the corresponding tiling strategy can be selected based on the shape during operator execution.

An operator can register with only one operator selector. To change the operator selector, call aclopUnregisterCompileFunc to unregister and then call aclopRegisterCompileFunc to register again.

Restrictions

None.
Prototype

```c
aclError aclopRegisterCompileFunc(const char *opType, aclopCompileFunc func)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>opType</td>
<td>Input</td>
<td>Operator type</td>
</tr>
<tr>
<td>func</td>
<td>Input</td>
<td>Callback function of the operator selector</td>
</tr>
</tbody>
</table>

```c
typedef aclError (*aclopCompileFunc)(int numInputs, const aclTensorDesc *const inputDesc[], int numOutputs, const aclTensorDesc *const outputDesc[], const aclopAttr *opAttr, aclopKernelDesc *aclopKernelDesc);
```

Returns

0 indicates success. Other values indicate failure.

7.9.2 aclopUnregisterCompileFunc

Description

Unregisters the operator selector in dynamic shape scenarios.

Prototype

```c
aclError aclopUnregisterCompileFunc(const char *opType)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>opType</td>
<td>Input</td>
<td>Operator type</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.9.3 aclopCreateKernel

Description

Creates the kernel function of an operator in dynamic shape scenarios for operator running.
Restrictions

None.

Prototype

```c
aclError aclopCreateKernel(const char *opType,
const char *kernelId,
const char *kernelName,
void *binData,
int binSize,
aclopEngineType enginetype,
aclDataDeallocator deallocator)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>opType</td>
<td>Input</td>
<td>Operator type</td>
</tr>
<tr>
<td>kernelId</td>
<td>Input</td>
<td>Kernel ID</td>
</tr>
<tr>
<td>kernelName</td>
<td>Input</td>
<td>Operator kernel name, which must be the same as the <code>kernelName</code> argument in the operator binary file.</td>
</tr>
<tr>
<td>binData</td>
<td>Input</td>
<td>Memory address of the operator kernel file</td>
</tr>
<tr>
<td>binSize</td>
<td>Input</td>
<td>Memory size of the operator kernel file, in bytes.</td>
</tr>
</tbody>
</table>
| enginetype  | Input        | Operator execution engine. This parameter is valid only when `compileFlag` of `aclopUpdateParams` is set to `ACL_COMPILE_SYS`. Example:
```c
typedef enum aclEngineType {
    ACL_ENGINE_SYS, // Set when the execution target is not concerned.
    ACL_ENGINE_AICORE, // Build the operator into an AI Core operator.
    ACL_ENGINE_VECTOR, // Build the operator into a Vector Core operator.
} aclopEngineType;
```
| deallocator | Input        | Callback function to release `binData` memory. The callback function is set based on the construction of `binData` by the caller. If the data is released by the caller, leave this parameter empty.

The function signature is as follows:
```c
typedef void (*aclDataDeallocator)(void *data, size_t length);
```

Returns

0 indicates success. Other values indicate failure.
7.9.4 aclopSetKernelArgs

Description

Sets tiling arguments and number of blocks for concurrent execution in dynamic shape scenarios.

Restrictions

None.

Prototype

```c
aclError aclopSetKernelArgs(aclopKernelDesc *kernelDesc,
   const char *kernelId,
   uint32_t blockDim,
   const void *args,
   uint32_t argSize)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kernelDesc</td>
<td>Input</td>
<td>Kernel description buffer, which is the pointer to the aclopKernelDesc type.</td>
</tr>
<tr>
<td>kernelId</td>
<td>Input</td>
<td>Kernel ID, which must be the same as that passed to the aclopCreateKernel call.</td>
</tr>
<tr>
<td>blockDim</td>
<td>Input</td>
<td>Number of blocks for kernel concurrent execution</td>
</tr>
<tr>
<td>args</td>
<td>Input</td>
<td>Tiling argument</td>
</tr>
<tr>
<td>argSize</td>
<td>Input</td>
<td>Tiling argument size, in bytes</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.9.5 aclopSetKernelWorkspaceSizes

Description

Sets the workspace parameters of an operator in dynamic shape scenarios.
Restrictions

This is an optional interface and can be selected as required.

Prototype

```c
aclError aclopSetKernelWorkspaceSizes(aclopKernelDesc *kernelDesc, int numWorkspaces, size_t *workspaceSizes)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kernelDesc</td>
<td>Input</td>
<td>Kernel description buffer, which is the pointer to the <code>aclopKernelDesc</code> type.</td>
</tr>
<tr>
<td>numWorkspaces</td>
<td>Input</td>
<td>Number of workspaces</td>
</tr>
<tr>
<td>workspaceSizes</td>
<td>Input</td>
<td>Array address of the workspace sizes</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.9.6 aclopUpdateParams

Description

Triggers the calling of operator selector based on the specified operator in dynamic shape scenarios.

Restrictions

None.

Prototype

```c
aclError aclopUpdateParams(const char *opType,
int numInputs,
const aclTensorDesc *const inputDesc[],
int numOutputs,
const aclTensorDesc *const outputDesc[],
const aclopAttr *attr)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>opType</td>
<td>Input</td>
<td>Operator type</td>
</tr>
<tr>
<td>numInputs</td>
<td>Input</td>
<td>Number of input tensors</td>
</tr>
<tr>
<td>inputDesc</td>
<td>Input</td>
<td>Input tensor description</td>
</tr>
<tr>
<td>numOutputs</td>
<td>Input</td>
<td>Number of output tensors</td>
</tr>
<tr>
<td>outputDesc</td>
<td>Input</td>
<td>Output tensor description</td>
</tr>
<tr>
<td>attr</td>
<td>Input</td>
<td>Operator attributes</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.9.7 aclopCompile

NOTICE

This API is defined in the /fwkaccllib/include/acl/acl_op_compiler.h file in the FwkACLlib installation path. This API is reserved for subsequent features and is not recommended currently.

Description

Builds an operator.

Restrictions

None.

Prototype

```c
aclError aclopCompile(const char *opType,
int numInputs,
const aclTensorDesc *const inputDesc[],
int numOutputs,
const aclTensorDesc *const outputDesc[],
const aclopAttr *attr,
aclopEngineType engineType,
aclopCompileType compileFlag,
```

const char* opPath)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>opType</td>
<td>Input</td>
<td>Operator type</td>
</tr>
<tr>
<td>numInputs</td>
<td>Input</td>
<td>Number of input tensors</td>
</tr>
<tr>
<td>inputDesc</td>
<td>Input</td>
<td>Input tensor description</td>
</tr>
<tr>
<td>numOutputs</td>
<td>Input</td>
<td>Number of output tensors</td>
</tr>
<tr>
<td>outputDesc</td>
<td>Input</td>
<td>Output tensor description</td>
</tr>
<tr>
<td>attr</td>
<td>Input</td>
<td>Operator attributes</td>
</tr>
<tr>
<td>engineType</td>
<td>Input</td>
<td>Operator execution target</td>
</tr>
<tr>
<td>compileFlag</td>
<td>Input</td>
<td>Operator building flag</td>
</tr>
<tr>
<td>opPath</td>
<td>Input</td>
<td>Operator path</td>
</tr>
</tbody>
</table>

typedef enum aclEngineType {
  ACL_ENGINE_SYS, // Set when the execution target is not concerned.
  ACL_ENGINE_AICORE, // Build the operator into an AI Core operator.
  ACL_ENGINE_VECTOR, // Build the operator into a Vector Core operator.
} aclOpEngineType;

typedef enum aclCompileType {
  ACL_COMPILE_SYS, // The operator has been registered with GE and FE
  ACL_COMPILE_UNREGISTERED // The operator has not been registered with GE and FE. The py source file needs to be used to build the operator. (This parameter is not supported)
} aclOpCompileType;

Returns

0 indicates success. Other values indicate failure.

7.10 Operator Loading and Execution

7.10.1 aclOpSetModelDir

Description

Sets the directory to store the model file, which is an *.om file built from a single operator. It is a synchronous interface.
Restrictions

- The `aclopSetModelDir` interface can be called only once in a process.
- The maximum number of operators that are being executed in a process is 60,000.

Prototype

```c
aclError aclopSetModelDir(const char *modelDir)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelDir</td>
<td>Input</td>
<td>Directory where the model file is located. Multiple levels of directories can be set. However, the system reads models in three levels down from the last level. For example, if <code>modelDir</code> is set to <code>dir0/dir1</code> and <code>dir1</code> contains <code>dir2/ dir3/dir4</code>, the system reads model files in <code>dir1</code>, <code>dir1/dir2</code> and <code>dir1/ dir2/dir3</code> only.</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.10.2 aclopLoad

Description

Loads the single-operator model data from the memory. The memory is managed by the user. It is a synchronous interface.

Single-operator model data refers to the data that is loaded to the memory from the `.om` file. The `.om` file is built from a single operator.

Restrictions

The maximum number of operators that are being executed in a process is 60,000.

Prototype

```c
aclError aclopLoad(const void *model, size_t modelSize)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>Input</td>
<td>Memory address of the single-operator model data</td>
</tr>
<tr>
<td>modelSize</td>
<td>Input</td>
<td>Size of the model data in the memory, in bytes</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.10.3 aclopExecute

Description

Executes an operator. It is an asynchronous interface.

Restrictions

The input and output organizations of each operator are different, and therefore, this interface needs to be called by the application in strict accordance with the input and output organizations of the operator.

When aclopExecute is called, the ACL searches for the corresponding task based on the optype, input tensor description, output tensor description, and attribute information, and delivers the task for execution.

Prototype

```c
aclError aclopExecute(const char *opType,
                     int numInputs,
                     const aclTensorDesc *const inputDesc[],
                     const aclDataBuffer *const inputs[],
                     int numOutputs,
                     const aclTensorDesc *const outputDesc[],
                     aclDataBuffer *const outputs[],
                     const aclopAttr *attr,
                     aclrtStream stream)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>opType</td>
<td>Input</td>
<td>Operator type</td>
</tr>
<tr>
<td>numInputs</td>
<td>Input</td>
<td>Number of input tensors</td>
</tr>
<tr>
<td>inputDesc</td>
<td>Input</td>
<td>Input tensor description</td>
</tr>
<tr>
<td>inputs</td>
<td>Input</td>
<td>Input tensors</td>
</tr>
<tr>
<td>numOutputs</td>
<td>Input</td>
<td>Number of output tensors</td>
</tr>
<tr>
<td>outputDesc</td>
<td>Input</td>
<td>Output tensor description</td>
</tr>
<tr>
<td>outputs</td>
<td>Output</td>
<td>Output tensors</td>
</tr>
<tr>
<td>attr</td>
<td>Input</td>
<td>Operator attributes</td>
</tr>
<tr>
<td>stream</td>
<td>Input</td>
<td>Target stream of the operator</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.10.4 aclopCompileAndExecute

NOTICE

This API is defined in the /fwkacllib/include/acl/acl_op_compiler.h file in the FwkACLlib installation path. This API is reserved for subsequent features and is not recommended currently.

Description

Builds and executes an operator. It is an asynchronous interface.

Restrictions

The input and output organizations of each operator are different, and therefore, this interface needs to be called by the application in strict accordance with the input and output organizations of the operator.

When aclopCompileAndExecute is called, the ACL searches for the corresponding task based on the optype, input tensor description, output tensor description, and operator attributes, and then builds and executes the operator.

Prototype

```c
aclError aclopCompileAndExecute(const char *opType,
                               int numInputs,
```
const aclTensorDesc *const inputDesc[],
const aclDataBuffer *const inputs[],
int numOutputs,
const aclTensorDesc *const outputDesc[],
aclDataBuffer *const outputs[],
const aclopAttr *attr,
aclopEngineType engineType,
aclopCompileType compileFlag,
const char *opPath,
aclrtStream stream);

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>opType</td>
<td>Input</td>
<td>Operator type</td>
</tr>
<tr>
<td>numInputs</td>
<td>Input</td>
<td>Number of input tensors</td>
</tr>
<tr>
<td>inputDesc</td>
<td>Input</td>
<td>Input tensor description</td>
</tr>
<tr>
<td>inputs</td>
<td>Input</td>
<td>Input tensors</td>
</tr>
<tr>
<td>numOutputs</td>
<td>Input</td>
<td>Number of output tensors</td>
</tr>
<tr>
<td>outputDesc</td>
<td>Input</td>
<td>Output tensor description</td>
</tr>
<tr>
<td>outputs</td>
<td>Output</td>
<td>Output tensors</td>
</tr>
<tr>
<td>attr</td>
<td>Input</td>
<td>Operator attributes</td>
</tr>
<tr>
<td>engineType</td>
<td>Input</td>
<td>Operator execution target</td>
</tr>
<tr>
<td>compileFlag</td>
<td>Input</td>
<td>Operator building flag</td>
</tr>
<tr>
<td>opPath</td>
<td>Input</td>
<td>Operator path</td>
</tr>
</tbody>
</table>

```c
typedef enum aclEngineType {
    ACL_ENGINE_SYS,    // Set when the execution
target is not concerned.
    ACL_ENGINE_AICORE, // Build the operator
    ACL_ENGINE_VECTOR, // Build the operator
} aclopEngineType;
```

```c
typedef enum aclCompileType {
    ACL_COMPILE_SYS,    // The operator has
    ACL_COMPILE_UNREGISTERED // The
operator has not been registered with GE and
FE. The py source file needs to be used to
build the operator. (This parameter is not
supported)
} aclopCompileType;
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stream</td>
<td>Input</td>
<td>Target stream of the operator</td>
</tr>
</tbody>
</table>

**Returns**

0 indicates success. Other values indicate failure.

### 7.10.5 aclopExecWithHandle

**Description**

Calls an operator in Handle mode with better performance. It is an asynchronous interface.

**Prototype**

```c
acError aclopExecWithHandle(aclopHandle *handle,
int numInputs,
const aclDataBuffer *const inputs[],
int numOutputs,
aclDataBuffer *const outputs[],
aclrtStream stream);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
</table>
| handle    | Input        | Handle of the operator to be executed
|           |              | `aclopCreateHandle` has been called in advance to create data of the `aclopHandle` type. |
| numInputs | Input        | Number of input tensors          |
| inputs    | Input        | Input tensor
|           |              | `aclCreateDataBuffer` has been called in advance to create data of the `aclDataBuffer` type. |
| numOutputs| Input        | Number of output tensors         |
| outputs   | Output       | Output tensor                    |
| stream    | Input        | Target stream of the operator    |
Returns

0 indicates success. Other values indicate failure.

7.11 CBLAS Interfaces

7.11.1 Shape and Memory Size Calculation of Matrices A, B, and C

- If the transpose flag of matrices A, B, and C is set to ACL_TRANS_N or ACL_TRANS_T, the size of the memory allocated to store the matrix data must match the actual data size. The formulas for calculating the shape and memory size are as follows:
  - Matrix A: shape = (m, k); Memory size = m * k * sizeof(dataTypeA)
  - Matrix B: shape = (k, n); Memory size = k * n * sizeof(dataTypeB)
  - Matrix C: shape = (m, n); Memory size = m * n * sizeof(dataTypeC)

- If the transpose flag of matrices A, B, and C is set to ACL_TRANS_NZ, the internal data format is used. The matrices are 4D. The formulas for calculating the shape and memory size are as follows (assuming that m, k, and n are the original axes):
  - When matrix A and matrix B are of type aclFloat16, m, k, and n should be rounded up to the nearest multiples of 16, respectively.
    - Matrix A: shape = (k/16, m/16, 16, 16); Memory size = m/16*16 * k/16*16 * sizeof(dataTypeA)
    - Matrix B: shape = (n/16, k/16, 16, 16); Memory size = k/16*16 * n/16*16 * sizeof(dataTypeB)
    - Matrix C: shape = (n/16, m/16, 16, 16); Memory size = m/16*16 * n/16*16 * sizeof(dataTypeC)
  - When matrix A and matrix B are of type int8_t, the axes to reduce should be rounded up to the nearest multiples of 32 and the axes not to reduce should be rounded up to the nearest multiples of 16, respectively.
    - Matrix A: shape = (k/32, m/16, 16, 32); Memory size = m/16*16 * k/32*32 * sizeof(dataTypeA)
    - Matrix B: shape = (k/32, n/16, 32, 16); Memory size = k/32*32 * n/16*16 * sizeof(dataTypeB)
    - Matrix C: shape = (n/16, m/16, 16, 16); Memory size = m/16*16 * n/16*16 * sizeof(dataTypeC)

NOTE

⌈⌉ indicates rounding-up.
7.11.2 aclblasGemvEx

Description

Computes matrix-vector multiplication. The data types of the input and output data are set using arguments. It is an asynchronous interface.

\[ y = \alpha Ax + \beta y \]

Restrictions

Only the following data types for \( A \), \( x \) and \( y \) are supported, and data type of \( \alpha \) and \( \beta \) are the same as \( y \).

<table>
<thead>
<tr>
<th>Data Type of A</th>
<th>Data Type of x</th>
<th>Data Type of y</th>
</tr>
</thead>
<tbody>
<tr>
<td>aclFloat16</td>
<td>aclFloat16</td>
<td>aclFloat16</td>
</tr>
<tr>
<td>aclFloat16</td>
<td>aclFloat16</td>
<td>float(float32)</td>
</tr>
<tr>
<td>int8_t</td>
<td>int8_t</td>
<td>float(float32)</td>
</tr>
<tr>
<td>int8_t</td>
<td>int8_t</td>
<td>int32_t</td>
</tr>
</tbody>
</table>

Prototype

```c
aclError aclblasGemvEx(aclTransType transA,
int m,
int n,
const void *alpha,
const void *a,
int lda,
aclDataType dataTypeA,
const void *x,
int incx,
aclDataType dataTypeX,
const void *beta,
void *y,
int incy,
aclDataType dataTypeY,
aclComputeType type,
aclrtStream stream);
```
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>transA</td>
<td>Input</td>
<td>Transpose flag of matrix A</td>
</tr>
<tr>
<td>m</td>
<td>Input</td>
<td>Number of rows in matrix A (row-major first in data storing)</td>
</tr>
<tr>
<td>n</td>
<td>Input</td>
<td>Number of columns in matrix A</td>
</tr>
<tr>
<td>alpha</td>
<td>Input</td>
<td>Data pointer to scalar α used for executing the multiplication operation</td>
</tr>
<tr>
<td>a</td>
<td>Input</td>
<td>Data pointer to matrix A</td>
</tr>
<tr>
<td>lda</td>
<td>Input</td>
<td>Leading dimension of matrix A. For row-major transpose, lda is the number of columns of matrix A. This parameter is reserved and can only be set to –1.</td>
</tr>
<tr>
<td>dataTypeA</td>
<td>Input</td>
<td>Data type of matrix A</td>
</tr>
<tr>
<td>x</td>
<td>Input</td>
<td>Data pointer to vector x</td>
</tr>
<tr>
<td>incx</td>
<td>Input</td>
<td>Increment between successive x elements. This parameter is reserved and can only be set to –1.</td>
</tr>
<tr>
<td>dataTypeX</td>
<td>Input</td>
<td>Data type of vector x</td>
</tr>
<tr>
<td>beta</td>
<td>Input</td>
<td>Data pointer to scalar β used for executing the multiplication operation</td>
</tr>
<tr>
<td>y</td>
<td>Input/Output</td>
<td>Data pointer to vector y</td>
</tr>
<tr>
<td>incy</td>
<td>Input</td>
<td>Increment between successive y elements. This parameter is reserved and can only be set to –1.</td>
</tr>
<tr>
<td>dataTypeY</td>
<td>Input</td>
<td>Data type of vector y</td>
</tr>
<tr>
<td>type</td>
<td>Input</td>
<td>Computation precision type (high precision by default)</td>
</tr>
<tr>
<td>stream</td>
<td>Input</td>
<td>Stream where the operator is executed</td>
</tr>
</tbody>
</table>

### Returns

0 indicates success. Other values indicate failure.
7.11.3 aclblasCreateHandleForGemvEx

**Description**

Creates a handle to matrix-vector multiplication. The data types of the input and output data are set using arguments. It is a synchronous interface.

**Restrictions**

Only the following data types for A, x and y are supported.

<table>
<thead>
<tr>
<th>Data Type of A</th>
<th>Data Type of x</th>
<th>Data Type of y</th>
</tr>
</thead>
<tbody>
<tr>
<td>aclFloat16</td>
<td>aclFloat16</td>
<td>aclFloat16</td>
</tr>
<tr>
<td>aclFloat16</td>
<td>aclFloat16</td>
<td>float(float32)</td>
</tr>
<tr>
<td>int8_t</td>
<td>int8_t</td>
<td>float(float32)</td>
</tr>
<tr>
<td>int8_t</td>
<td>int8_t</td>
<td>int32_t</td>
</tr>
</tbody>
</table>

**Prototype**

```c
aclError aclblasCreateHandleForGemvEx(aclTransType transA,
                                   int m,
                                   int n,
                                   aclDataType dataTypeA,
                                   aclDataType dataTypeX,
                                   aclDataType dataTypeY,
                                   aclComputeType type,
                                   aclopHandle **handle)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>transA</td>
<td>Input</td>
<td>Transpose flag of matrix A</td>
</tr>
<tr>
<td>m</td>
<td>Input</td>
<td>Number of rows in matrix A (row-major first in data storing)</td>
</tr>
<tr>
<td>n</td>
<td>Input</td>
<td>Number of columns in matrix A</td>
</tr>
<tr>
<td>dataTypeA</td>
<td>Input</td>
<td>Data type of matrix A</td>
</tr>
<tr>
<td>dataTypeX</td>
<td>Input</td>
<td>Data type of vector x</td>
</tr>
<tr>
<td>dataTypeY</td>
<td>Input</td>
<td>Data type of vector y</td>
</tr>
</tbody>
</table>
### Parameter Input/Output Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Input</td>
<td>Computation precision type (high precision by default)</td>
</tr>
<tr>
<td>handle</td>
<td>Output</td>
<td>Pointer to the pointer to the handle data of the operator executed</td>
</tr>
</tbody>
</table>

### Returns

0 indicates success. Other values indicate failure.

### 7.11.4 aclblasHgemv

#### Description

Computes matrix-vector multiplication. The input and output data type is aclFloat16. It is an asynchronous interface.

\[ y = \alpha Ax + \beta y \]

#### Prototype

```c
aclError aclblasHgemv(aclTransType transA, int m, int n, const aclFloat16 *alpha, const aclFloat16 *a, int lda, const aclFloat16 *x, int incx, const aclFloat16 *beta, aclFloat16 *y, int incy, aclComputeType type, aclrtStream stream)
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>transA</td>
<td>Input</td>
<td>Transpose flag of matrix A</td>
</tr>
<tr>
<td>Parameter</td>
<td>Input/Output</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>m</td>
<td>Input</td>
<td>Number of rows in matrix A (row-major first in data storing)</td>
</tr>
<tr>
<td>n</td>
<td>Input</td>
<td>Number of columns in matrix A</td>
</tr>
<tr>
<td>alpha</td>
<td>Input</td>
<td>Scalar α used for executing the multiplication operation</td>
</tr>
<tr>
<td>a</td>
<td>Input</td>
<td>Data pointer to matrix A</td>
</tr>
<tr>
<td>lda</td>
<td>Input</td>
<td>Leading dimension of matrix A. For row-major transpose, lda is the number of columns of matrix A. This parameter is reserved and can only be set to −1.</td>
</tr>
<tr>
<td>x</td>
<td>Input</td>
<td>Data pointer to vector x</td>
</tr>
<tr>
<td>incx</td>
<td>Input</td>
<td>Increment between successive x elements. This parameter is reserved and can only be set to −1.</td>
</tr>
<tr>
<td>beta</td>
<td>Input</td>
<td>Scalar β used for executing the multiplication operation.</td>
</tr>
<tr>
<td>y</td>
<td>Input/Output</td>
<td>Data pointer to vector y</td>
</tr>
<tr>
<td>incy</td>
<td>Input</td>
<td>Increment between successive y elements. This parameter is reserved and can only be set to −1.</td>
</tr>
<tr>
<td>type</td>
<td>Input</td>
<td>Computation precision type (high precision by default)</td>
</tr>
<tr>
<td>stream</td>
<td>Input</td>
<td>Stream where the operator is executed</td>
</tr>
</tbody>
</table>

**Returns**

0 indicates success. Other values indicate failure.

**7.11.5 aclblasCreateHandleForHgemv**

**Description**

Creates a handle to matrix-vector multiplication. The data type of the input and output data is aclFloat16. It is a synchronous interface.
Prototype

```c
aclError aclblasCreateHandleForHgemv(aclTransType transA,
int m,
int n,
aclopComputeType type,
aclopHandle **handle)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>transA</td>
<td>Input</td>
<td>Transpose flag of matrix A</td>
</tr>
<tr>
<td>m</td>
<td>Input</td>
<td>Number of rows in matrix A (row-major first in data storing)</td>
</tr>
<tr>
<td>n</td>
<td>Input</td>
<td>Number of columns in matrix A</td>
</tr>
<tr>
<td>type</td>
<td>Input</td>
<td>Computation precision type (high precision by default)</td>
</tr>
<tr>
<td>handle</td>
<td>Output</td>
<td>Pointer to the pointer to the handle data of the operator executed</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.11.6 aclblasS8gemv

Description

Computes matrix-vector multiplication. The input data type is `int8_t`, and the output data type is `int32_t`. It is an asynchronous interface.

\[ y = \alpha Ax + \beta y \]

Prototype

```c
aclError aclblasS8gemv(aclTransType transA,
int m,
int n,
const int32_t *alpha,
const int8_t *a,
int lda,
const int8_t *x,
```
int incx,
const int32_t *beta,
int32_t *y,
int incy,

aclComputeType type,
aclrtStream stream)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>transA</td>
<td>Input</td>
<td>Transpose flag of matrix A</td>
</tr>
<tr>
<td>m</td>
<td>Input</td>
<td>Number of rows in matrix A (row-major first in data storing)</td>
</tr>
<tr>
<td>n</td>
<td>Input</td>
<td>Number of columns in matrix A</td>
</tr>
<tr>
<td>alpha</td>
<td>Input</td>
<td>Scalar α used for executing the multiplication operation</td>
</tr>
<tr>
<td>a</td>
<td>Input</td>
<td>Data pointer to matrix A</td>
</tr>
<tr>
<td>lda</td>
<td>Input</td>
<td>Leading dimension of matrix A. For row-major transpose, lda is the number of columns of matrix A. This parameter is reserved and can only be set to -1.</td>
</tr>
<tr>
<td>x</td>
<td>Input</td>
<td>Data pointer to vector x</td>
</tr>
<tr>
<td>incx</td>
<td>Input</td>
<td>Increment between successive x elements</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This parameter is reserved and can only be set to -1.</td>
</tr>
<tr>
<td>beta</td>
<td>Input</td>
<td>Scalar β used for executing the multiplication operation</td>
</tr>
<tr>
<td>y</td>
<td>Input/Output</td>
<td>Data pointer to vector y</td>
</tr>
<tr>
<td>incy</td>
<td>Input</td>
<td>Increment between successive y elements</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This parameter is reserved and can only be set to -1.</td>
</tr>
<tr>
<td>type</td>
<td>Input</td>
<td>Computation precision type (high precision by default)</td>
</tr>
<tr>
<td>stream</td>
<td>Input</td>
<td>Stream where the operator is executed</td>
</tr>
</tbody>
</table>
Returns

0 indicates success. Other values indicate failure.

7.11.7 aclblasCreateHandleForS8gemv

Description

Creates a handle to matrix-vector multiplication. The input data type is int8_t, and the output data type is int32_t. It is a synchronous interface.

Prototype

```c
aclError aclblasCreateHandleForS8gemv(aclTransType transA,
int m,
int n,
aclComputeType type,
aclOpHandle **handle)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>transA</td>
<td>Input</td>
<td>Transpose flag of matrix A</td>
</tr>
<tr>
<td>m</td>
<td>Input</td>
<td>Number of rows in matrix A (row-major first in data storing)</td>
</tr>
<tr>
<td>n</td>
<td>Input</td>
<td>Number of columns in matrix A</td>
</tr>
<tr>
<td>type</td>
<td>Input</td>
<td>Computation precision type (high precision by default)</td>
</tr>
<tr>
<td>handle</td>
<td>Output</td>
<td>Pointer to the pointer to the handle data of the operator executed</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.11.8 aclblasGemmEx

Description

Computes matrix-matrix multiplication. The data types of the input and output data are set using arguments. It is an asynchronous interface.

\[ C = \alpha AB + \beta C \]
Restrictions

Only the following data types for A, B and C are supported, and data type of \( \alpha \) and \( \beta \) are the same as C.

<table>
<thead>
<tr>
<th>Data Type of A</th>
<th>Data Type of B</th>
<th>Data Type of C</th>
</tr>
</thead>
<tbody>
<tr>
<td>aclFloat16</td>
<td>aclFloat16</td>
<td>aclFloat16</td>
</tr>
<tr>
<td>aclFloat16</td>
<td>aclFloat16</td>
<td>float(float32)</td>
</tr>
<tr>
<td>int8_t</td>
<td>int8_t</td>
<td>float(float32)</td>
</tr>
<tr>
<td>int8_t</td>
<td>int8_t</td>
<td>int32_t</td>
</tr>
</tbody>
</table>

Prototype

```c
 aclError aclblasGemmEx(aclTransType transA,
                        aclTransType transB,
                        aclTransType transC,
                        int m,
                        int n,
                        int k,
                        const void *alpha,
                        const void *matrixA,
                        int lda,
                        aclDataType dataTypeA,
                        const void *matrixB,
                        int ldb,
                        aclDataType dataTypeB,
                        const void *beta,
                        void *matrixC,
                        int ldc,
                        aclDataType dataTypeC,
                        aclComputeType type,
                        aclrtStream stream)
```
## Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>transA</td>
<td>Input</td>
<td>Transpose flag of matrix A</td>
</tr>
<tr>
<td>transB</td>
<td>Input</td>
<td>Transpose flag of matrix B</td>
</tr>
<tr>
<td>transC</td>
<td>Input</td>
<td>Transpose flag of matrix C. Currently, only <code>ACL_TRANS_N</code> is supported.</td>
</tr>
<tr>
<td>m</td>
<td>Input</td>
<td>Number of rows in matrix A and C</td>
</tr>
<tr>
<td>n</td>
<td>Input</td>
<td>Number of columns in matrix B and C</td>
</tr>
<tr>
<td>k</td>
<td>Input</td>
<td>Number of columns in matrix A and number of rows in matrix B</td>
</tr>
<tr>
<td>alpha</td>
<td>Input</td>
<td>Data pointer to scalar α used for executing the multiplication operation</td>
</tr>
<tr>
<td>matrixA</td>
<td>Input</td>
<td>Data pointer to matrix A</td>
</tr>
<tr>
<td>lda</td>
<td>Input</td>
<td>Leading dimension of matrix A. For row-major transpose, <code>lda</code> is the number of columns of matrix A. This parameter is reserved and can only be set to –1.</td>
</tr>
<tr>
<td>dataTypeA</td>
<td>Input</td>
<td>Data type of matrix A</td>
</tr>
<tr>
<td>matrixB</td>
<td>Input</td>
<td>Data pointer to matrix B</td>
</tr>
<tr>
<td>ldb</td>
<td>Input</td>
<td>Leading dimension of matrix B. For row-major transpose, <code>ldb</code> is the number of columns of matrix B. This parameter is reserved and can only be set to –1.</td>
</tr>
<tr>
<td>dataTypeB</td>
<td>Input</td>
<td>Data type of matrix B</td>
</tr>
<tr>
<td>beta</td>
<td>Input</td>
<td>Data pointer to scalar β used for executing the multiplication operation</td>
</tr>
<tr>
<td>matrixC</td>
<td>Input/Output</td>
<td>Data pointer to matrix C</td>
</tr>
<tr>
<td>ldc</td>
<td>Input</td>
<td>Leading dimension of matrix C. This parameter is reserved and can only be set to –1.</td>
</tr>
<tr>
<td>dataTypeC</td>
<td>Input</td>
<td>Data type of matrix C</td>
</tr>
<tr>
<td>type</td>
<td>Input</td>
<td>Computation precision type (high precision by default)</td>
</tr>
<tr>
<td>Parameter</td>
<td>Input/Output</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>stream</td>
<td>Input</td>
<td>Stream where the operator is executed</td>
</tr>
</tbody>
</table>

**Returns**

0 indicates success. Other values indicate failure.

### 7.11.9 aclblasCreateHandleForGemmEx

**Description**

Creates a handle to matrix-matrix multiplication. The data types of the input and output data are set using arguments. It is a synchronous interface.

**Restrictions**

Only the following data types for A, B and C are supported.

<table>
<thead>
<tr>
<th>Data Type of A</th>
<th>Data Type of B</th>
<th>Data Type of C</th>
</tr>
</thead>
<tbody>
<tr>
<td>aclFloat16</td>
<td>aclFloat16</td>
<td>aclFloat16</td>
</tr>
<tr>
<td>aclFloat16</td>
<td>aclFloat16</td>
<td>float(float32)</td>
</tr>
<tr>
<td>int8_t</td>
<td>int8_t</td>
<td>float(float32)</td>
</tr>
<tr>
<td>int8_t</td>
<td>int8_t</td>
<td>int32_t</td>
</tr>
</tbody>
</table>

**Prototype**

```c
aclError aclblasCreateHandleForGemmEx(aclTransType transA,
                                      aclTransType transB,
                                      aclTransType transC,
                                      int m,
                                      int n,
                                      int k,
                                      aclDataType dataTypeA,
                                      aclDataType dataTypeB,
                                      aclDataType dataTypeC,
                                      aclComputeType type,
                                      aclopHandle **handle)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>transA</td>
<td>Input</td>
<td>Transpose flag of matrix A</td>
</tr>
<tr>
<td>transB</td>
<td>Input</td>
<td>Transpose flag of matrix B</td>
</tr>
<tr>
<td>transC</td>
<td>Input</td>
<td>Transpose flag of matrix C. Currently, only ACL_TRANS_N is supported.</td>
</tr>
<tr>
<td>m</td>
<td>Input</td>
<td>Number of rows in matrix A and C</td>
</tr>
<tr>
<td>n</td>
<td>Input</td>
<td>Number of columns in matrix B and C</td>
</tr>
<tr>
<td>k</td>
<td>Input</td>
<td>Number of columns in matrix A and number of rows in matrix B</td>
</tr>
<tr>
<td>dataTypeA</td>
<td>Input</td>
<td>Data type of matrix A</td>
</tr>
<tr>
<td>dataTypeB</td>
<td>Input</td>
<td>Data type of matrix B</td>
</tr>
<tr>
<td>dataTypeC</td>
<td>Input</td>
<td>Data type of matrix C</td>
</tr>
<tr>
<td>type</td>
<td>Input</td>
<td>Computation precision type (high precision by default)</td>
</tr>
<tr>
<td>handle</td>
<td>Output</td>
<td>Pointer to the pointer to the handle data of the operator executed</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.11.10 aclblasHgemm

Description

Computes matrix-matrix multiplication. The input and output data type is aclFloat16. It is an asynchronous interface.

\[ C = \alpha AB + \beta C \]

Prototype

\[
\text{aclError aclblasHgemm(aclTransType transA, aclTransType transB, aclTransType transC, int m, int n,}
\]
int k,
const aclFloat16 *alpha,
const aclFloat16 *matrixA,
int lda,
const aclFloat16 *matrixB,
int ldb,
const aclFloat16 *beta,
aclFloat16 *matrixC,
int ldc,
aclComputeType type,
aclrtStream stream)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>transA</td>
<td>Input</td>
<td>Transpose flag of matrix A</td>
</tr>
<tr>
<td>transB</td>
<td>Input</td>
<td>Transpose flag of matrix B</td>
</tr>
<tr>
<td>transC</td>
<td>Input</td>
<td>Transpose flag of matrix C. Currently, only ACL_TRANS_N is supported.</td>
</tr>
<tr>
<td>m</td>
<td>Input</td>
<td>Number of rows in matrix A and C</td>
</tr>
<tr>
<td>n</td>
<td>Input</td>
<td>Number of columns in matrix B and C</td>
</tr>
<tr>
<td>k</td>
<td>Input</td>
<td>Number of columns in matrix A and number of rows in matrix B</td>
</tr>
<tr>
<td>alpha</td>
<td>Input</td>
<td>Scalar α used for executing the multiplication operation</td>
</tr>
<tr>
<td>matrixA</td>
<td>Input</td>
<td>Data pointer to matrix A</td>
</tr>
<tr>
<td>lda</td>
<td>Input</td>
<td>Leading dimension of matrix A. For row-major transpose, lda is the number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>of columns of matrix A. This parameter is reserved and can only be set to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-1.</td>
</tr>
<tr>
<td>matrixB</td>
<td>Input</td>
<td>Data pointer to matrix B</td>
</tr>
<tr>
<td>Parameter</td>
<td>Input/Output</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>ldb</td>
<td>Input</td>
<td>Leading dimension of matrix B. For row-major transpose, ldb is the number of columns of matrix B. This parameter is reserved and can only be set to -1.</td>
</tr>
<tr>
<td>beta</td>
<td>Input</td>
<td>Scalar β used for executing the multiplication operation</td>
</tr>
<tr>
<td>matrixC</td>
<td>Input/Output</td>
<td>Data pointer to matrix C</td>
</tr>
<tr>
<td>ldc</td>
<td>Input</td>
<td>Leading dimension of matrix C. This parameter is reserved and can only be set to -1.</td>
</tr>
<tr>
<td>type</td>
<td>Input</td>
<td>Computation precision type (high precision by default)</td>
</tr>
<tr>
<td>stream</td>
<td>Input</td>
<td>Stream where the operator is executed</td>
</tr>
</tbody>
</table>

**Returns**

0 indicates success. Other values indicate failure.

### 7.11.11 aclblasCreateHandleForHgemm

**Description**

Creates a handle to matrix-matrix multiplication. The data type of the input and output data is aclFloat16. It is a synchronous interface.

**Prototype**

```c
aclError aclblasCreateHandleForHgemm(aclTransType transA,
                                    aclTransType transB,
                                    aclTransType transC,
                                    int m,
                                    int n,
                                    int k,
                                    aclComputeType type,
                                    aclopHandle **handle)
```
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>transA</td>
<td>Input</td>
<td>Transpose flag of matrix A</td>
</tr>
<tr>
<td>transB</td>
<td>Input</td>
<td>Transpose flag of matrix B</td>
</tr>
<tr>
<td>transC</td>
<td>Input</td>
<td>Transpose flag of matrix C. Currently, only <code>ACL_TRANS_N</code> is supported.</td>
</tr>
<tr>
<td>m</td>
<td>Input</td>
<td>Number of rows in matrix A and C</td>
</tr>
<tr>
<td>n</td>
<td>Input</td>
<td>Number of columns in matrix B and C</td>
</tr>
<tr>
<td>k</td>
<td>Input</td>
<td>Number of columns in matrix A and number of rows in matrix B</td>
</tr>
<tr>
<td>type</td>
<td>Input</td>
<td>Computation precision type (high precision by default)</td>
</tr>
<tr>
<td>handle</td>
<td>Output</td>
<td>Pointer to the pointer to the handle data of the operator executed</td>
</tr>
</tbody>
</table>

### Returns

0 indicates success. Other values indicate failure.

### 7.11.12 aclblasS8gemm

#### Description

Computes matrix-matrix multiplication. The input data type is `int8_t`, and the output data type is `int32_t`. It is an asynchronous interface.

\[ C = \alpha AB + \beta C \]

#### Prototype

```c
aclError aclblasS8gemm(aclTransType transA,
                        aclTransType transB,
                        aclTransType transC,
                        int m,
                        int n,
                        int k,
                        const int32_t *alpha,
                        const int8_t *matrixA,
```

---

Issue 01 (2020-07-30)  Copyright © Huawei Technologies Co., Ltd.  194
int lda,
const int8_t *matrixB,
int ldb,
const int32_t *beta,
int32_t *matrixC,
int ldc,
aclComputeType type,
aclrtStream stream)

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>transA</td>
<td>Input</td>
<td>Transpose flag of matrix A</td>
</tr>
<tr>
<td>transB</td>
<td>Input</td>
<td>Transpose flag of matrix B</td>
</tr>
<tr>
<td>transC</td>
<td>Input</td>
<td>Transpose flag of matrix C. Currently, only ACL_TRANS_N is supported.</td>
</tr>
<tr>
<td>m</td>
<td>Input</td>
<td>Number of rows in matrix A and C</td>
</tr>
<tr>
<td>n</td>
<td>Input</td>
<td>Number of columns in matrix B and C</td>
</tr>
<tr>
<td>k</td>
<td>Input</td>
<td>Number of columns in matrix A and number of rows in matrix B</td>
</tr>
<tr>
<td>alpha</td>
<td>Input</td>
<td>Scalar α used for executing the multiplication operation</td>
</tr>
<tr>
<td>matrixA</td>
<td>Input</td>
<td>Data pointer to matrix A</td>
</tr>
<tr>
<td>lda</td>
<td>Input</td>
<td>Leading dimension of matrix A. For row-major transpose, lda is the number of columns of matrix A. This parameter is reserved and can only be set to –1.</td>
</tr>
<tr>
<td>matrixB</td>
<td>Input</td>
<td>Data pointer to matrix B</td>
</tr>
<tr>
<td>ldb</td>
<td>Input</td>
<td>Leading dimension of matrix B. For row-major transpose, ldb is the number of columns of matrix B. This parameter is reserved and can only be set to –1.</td>
</tr>
<tr>
<td>beta</td>
<td>Input</td>
<td>Scalar β used for executing the multiplication operation</td>
</tr>
<tr>
<td>matrixC</td>
<td>Input/Output</td>
<td>Data pointer to matrix C</td>
</tr>
</tbody>
</table>
### Parameter Table

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldc</td>
<td>Input</td>
<td>Leading dimension of matrix C. This parameter is reserved and can only be set to -1.</td>
</tr>
<tr>
<td>type</td>
<td>Input</td>
<td>Computation precision type (high precision by default)</td>
</tr>
<tr>
<td>stream</td>
<td>Input</td>
<td>Stream where the operator is executed</td>
</tr>
</tbody>
</table>

### Returns

0 indicates success. Other values indicate failure.

### 7.11.13 aclblasCreateHandleForS8gemm

**Description**

Creates a handle to matrix-matrix multiplication. The input data type is int8_t, and the output data type is int32_t. It is a synchronous interface.

**Prototype**

```c
aclError aclblasCreateHandleForS8gemm(aclTransType transA,
                                   aclTransType transB,
                                   aclTransType transC,
                                   int m,
                                   int n,
                                   int k,
                                   aclComputeType type,
                                   aclopHandle **handle)
```

### Parameters Table

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>transA</td>
<td>Input</td>
<td>Transpose flag of matrix A</td>
</tr>
<tr>
<td>transB</td>
<td>Input</td>
<td>Transpose flag of matrix B</td>
</tr>
<tr>
<td>transC</td>
<td>Input</td>
<td>Transpose flag of matrix C. Currently, only ACL_TRANS_N is supported.</td>
</tr>
<tr>
<td>m</td>
<td>Input</td>
<td>Number of rows in matrix A and C</td>
</tr>
<tr>
<td>Parameter</td>
<td>Input/Output</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>n</td>
<td>Input</td>
<td>Number of columns in matrix B and C</td>
</tr>
<tr>
<td>k</td>
<td>Input</td>
<td>Number of columns in matrix A and number of rows in matrix B</td>
</tr>
<tr>
<td>type</td>
<td>Input</td>
<td>Computation precision type (high precision by default)</td>
</tr>
<tr>
<td>handle</td>
<td>Output</td>
<td>Pointer to the pointer to the handle data of the operator executed</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.11.14 aclopCast

Description

Converts the data type of the input tensor. It is an asynchronous interface.

Prototype

```c
aclError aclopCast(const aclTensorDesc *srcDesc,
const aclDataBuffer *srcBuffer,
const aclTensorDesc *dstDesc,
 aclDataBuffer *dstBuffer,
uint8_t truncate,
aclrtStream stream)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcDesc</td>
<td>Input</td>
<td>Input tensor description</td>
</tr>
<tr>
<td>srcBuffer</td>
<td>Input</td>
<td>Input tensor</td>
</tr>
<tr>
<td>dstDesc</td>
<td>Input</td>
<td>Output tensor description</td>
</tr>
<tr>
<td>dstBuffer</td>
<td>Output</td>
<td>Output tensor</td>
</tr>
<tr>
<td>truncate</td>
<td>Input</td>
<td>Reserved</td>
</tr>
<tr>
<td>stream</td>
<td>Input</td>
<td>Stream where the operator is executed</td>
</tr>
</tbody>
</table>
Returns

0 indicates success. Other values indicate failure.

7.11.15 aclopCreateHandleForCast

Description

Creates a handle to data type conversion. It is a synchronous interface.

Prototype

```c
aclError aclopCreateHandleForCast(aclTensorDesc *srcDesc,
                                 aclTensorDesc *dstDesc,
                                 uint8_t truncate,
                                 aclopHandle **handle)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcDesc</td>
<td>Input</td>
<td>Input tensor description</td>
</tr>
<tr>
<td>dstDesc</td>
<td>Input</td>
<td>Output tensor description</td>
</tr>
<tr>
<td>truncate</td>
<td>Input</td>
<td>Reserved</td>
</tr>
<tr>
<td>handle</td>
<td>Input</td>
<td>Pointer to the pointer to the handle data of the operator executed</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.12 Media Data Processing

Media data processing includes image decoding, cropping and resizing, and video decoding.

7.12.1 General Description

- About asynchronous interfaces
  For the asynchronous interfaces described in this chapter, a successful interface call only indicates the success of the task delivery, regardless of the execution result. For dependent interfaces, you are advised to specify the same stream for multiple interfaces to ensure tasks execution sequence, because tasks in the same stream are executed in accordance with the interfaces calling sequence.
When asynchronous APIs are called to decode, crop, and resize images, if tasks depend on each other, call `aclrtSynchronizeStream` to ensure that tasks in the stream are executed in order.

After an asynchronous API is called, resources cannot be released immediately. You need to call the synchronous wait API (for example, `aclrtSynchronizeStream`) to confirm that the tasks on the device side have been completed.

- **About memory allocation and release**
  
  If memory on the device is needed to store the input or output data before implementing the VPC, JPEGD, and JEPGE functions for media data processing, `acldvppMalloc` can be called to allocate memory, and `acldvppFree` to free memory.

- **The following media data processing functions are provided:**
  
  - Vision preprocessing core (VPC): supports image cropping, resizing, overlaying, stitching, and format conversion. For details, see 7.12.4.1 Functions and Restrictions.
  - JPEG decoder (JPEGD): decodes JPG, JPEG, JPG, and JPEG images into YUV images. For details, see 7.12.5.1 Functions and Restrictions.
  - Video encoder (JPEGE): encodes YUV images into JPG images. For details, see 7.12.6.1 Functions and Restrictions.
  - Video decoder (VDEC): decodes videos. For details, see 7.12.7.1 Functions and Restrictions.
  - Video encoder (VENC): encodes videos. For details, see 7.12.8.1 Functions and Restrictions.

- **Except `acldvppMalloc` and `acldvppFree`, other APIs described in the media data processing chapter can be called only on the host side.**

### 7.12.2 Memory Allocation and Free

#### 7.12.2.1 `acldvppMalloc`

**Description**

Allocates memory for media processing on the device. The allocated huge memory page meets the data processing requirements (for example, the start address is 128-byte aligned). It is a synchronous interface. `acldvppMalloc` and `acldvppFree` must be used in pair. Memory allocated using `acldvppMalloc` can only be freed by calling `acldvppFree`.

- The allocated memory size is the input size rounded up to the nearest multiple of 32, plus 32 bytes.
**NOTE**

If the huge memory allocated by calling `aclDvppMalloc` is divided and managed by the user, the user needs to round the actual data size of each image to the nearest multiple of 32, plus 32 bytes (ALIGN_UP_[len] + 32 bytes) for memory management.

For example, if the user needs to manage memory of n images using `aclDvppMalloc`, and the size of each image is `len` bytes, then the memory should be managed in accordance with the following formula: n x (ALIGN_UP_[len] + 32 bytes), and the memory address of each image is offset by (ALIGN_UP_[len] + 32 bytes).

ALIGN_UP indicates 32-byte upwards alignment as in this formula: ((len – 1)/32 + 1) x 32.

- If the allocation of huge memory by calling this interface fails, it only indicates that the huge memory in the system is insufficient.

**Restrictions**

Performance deterioration will be caused by the frequent calling of `aclDvppMalloc` to allocate memory and `aclDvppFree` to free memory. You are advised to allocate or manage memory in advance to avoid frequent memory application and freeing.

**Prototype**

```c
aclError aclDvppMalloc(void **devPtr, size_t size)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>devPtr</td>
<td>Output</td>
<td>Pointer to the pointer to the allocated memory on the device</td>
</tr>
<tr>
<td>size</td>
<td>Input</td>
<td>Allocated memory size, in bytes</td>
</tr>
</tbody>
</table>

**Returns**

0 indicates success. Other values indicate failure.

**7.12.2.2 aclDvppFree**

**Description**

Frees the memory allocated using `aclDvppMalloc`. It is a synchronous interface.

**Prototype**

```c
aclError aclDvppFree(void *devPtr)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>devPtr</td>
<td>Input</td>
<td>Pointer to the memory to be freed</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.12.3 Channel Creation and Destruction

7.12.3.1 VPC/JPEGD/JPEGE Image Processing Channel

7.12.3.1.1 aclvppCreateChannel

Description

Creates a channel for image data processing. A channel can be reused. After being destroyed, the channel is no longer available. It is a synchronous interface.

Restrictions

The channel is non-thread-safe, that is, a separate channel should be created for each thread.

Prototype

aclError aclvppCreateChannel(acldvppChannelDesc *channelDesc)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
</table>
| channelDesc | Input/Output | Channel description
Data of the acldvppChannelDesc type is created by calling aclvppCreateChannelDesc. |

Returns

0 indicates success. Other values indicate failure.
7.12.3.1.2 aclvppDestroyChannel

**Description**

Destroys a channel created by calling `aclvppCreateChannel` for image data processing. It is a synchronous interface.

**Prototype**

```c
aclError aclvppDestroyChannel(aclvppChannelDesc *channelDesc)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelDesc</td>
<td>Input</td>
<td>Channel description, which must be the same as the value of <code>channelDesc</code> specified in the <code>aclvppCreateChannel</code> call.</td>
</tr>
</tbody>
</table>

**Returns**

0 indicates success. Other values indicate failure.

7.12.3.2 VDEC Channel

7.12.3.2.1 aclvdecCreateChannel

**Description**

Creates a video decoding channel. A channel can be reused. After being destroyed, the channel is no longer available. It is a synchronous interface.

`aclvdecCreateChannel` encapsulates the `aclrtCreateStream` to explicitly create a stream, and `aclrtSubscribeReport` to specify a thread to process the callback function in the stream. The callback function and thread are specified in the calls to the `aclvdecSetChannelDesc` series. When implementing the VDEC function, `aclrtCreateStream` and `aclrtSubscribeReport` do not need to be called separately.

**Prototype**

```c
aclError aclvdecCreateChannel (aclvdecChannelDesc *channelDesc)
```
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelDesc</td>
<td>Input/Output</td>
<td>Channel description&lt;br&gt;Create data of the aclvdecChannelDesc type by calling aclvdecCreateChannelDesc, and set the channel description by calling aclvdecSetChannelDesc.</td>
</tr>
</tbody>
</table>

### Returns

0 indicates success. Other values indicate failure.

#### 7.12.3.2.2 aclvdecDestroyChannel

**Description**

Destroys a video decoding channel created by aclvdecCreateChannel. It is a synchronous interface.

The channel is destroyed after the transmitted frames are decoded and the callback function is handled.

aclvdecDestroyChannel encapsulates the aclrtUnSubscribeReport to unsubscribe a thread (callback function in the stream is no longer processed by the specified thread), and aclrtDestroyStream to destroy a stream. When implementing the VDEC function, aclrtUnSubscribeReport and aclrtDestroyStream do not need to be called separately.

**Prototype**

```c
aclError aclvdecDestroyChannel (aclvdecChannelDesc *channelDesc)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelDesc</td>
<td>Input</td>
<td>Channel description, which must be the same as the channelDesc argument passed to the aclvdecCreateChannel call.</td>
</tr>
</tbody>
</table>

### Returns

0 indicates success. Other values indicate failure.

#### 7.12.3.3 VENC Channel
7.12.3.3.1 aclvencCreateChannel

Description

Creates a video encoding channel. A channel can be reused. After being destroyed, the channel is no longer available. It is a synchronous interface.

Prototype

allError aclvencCreateChannel(aclvencChannelDesc *channelDesc)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
</table>
| channelDesc | Input/Output | Channel description  
Create data of the aclvencChannelDesc type by calling aclvencCreateChannelDesc, and set the channel description by calling aclvencSetChannelDesc.

Returns

0 indicates success. Other values indicate failure.

7.12.3.3.2 aclvencDestroyChannel

Description

Destroys a video encoding channel created by aclvencCreateChannel. It is a synchronous interface.

The channel is destroyed after the callback function is handled.

Prototype

allError aclvencDestroyChannel(aclvencChannelDesc *channelDesc)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelDesc</td>
<td>Input</td>
<td>Channel description, which must be the same as the channelDesc argument passed to the aclvencCreateChannel call.</td>
</tr>
</tbody>
</table>
Returns

0 indicates success. Other values indicate failure.

7.12.4 VPC

7.12.4.1 Functions and Restrictions

Description
The vision preprocessing core (VPC) provides the following functions:

- **Cropping**: crops one or more areas out of the input image.
- **Resizing**
  - VPC can process images with different resolutions:
    - Non-8K resizing: \( \text{widthStride} \in [32, 4096] \) and \( \text{heightStride} \in [6, 4096] \). The value range of \( \text{widthStride} \) varies according to the input image format. For details, see Table 7-2.
    - 8K resizing: \( \text{widthStride} \in (4096, 8192] \) or \( \text{heightStride} \in (4096, 8192] \)
  - Single-image cropping/resizing (supporting the uncompressed format) and single-image multi-ROI cropping/resizing (supporting the uncompressed format)
  - Other resizing modes, for example, original image resizing
- **Pasting**: crops an image out of an input image, resizes the cropped image, and pastes it in a specified area of the canvas. The canvas may be a blank image (when the output buffer allocated by the user is empty) or an existing image (when an image has been read into the output buffer allocated by the user). Note that the pasting concept here refers only to the case when the canvas is an existing image.
- **Stitching**: crops multiple images out of an input image, resizes the cropped images, and place them in a specified area of the output image.
- **Format conversion**
  - Converts between RGB and YUV. For details about the input and output image formats, see Table 7-2.
  - Converts a color image to a grayscale image. For the output image, only the data of the Y component is used.

Restrictions

- VPC has some restrictions on the input and output.
<table>
<thead>
<tr>
<th>VPC Input/Output</th>
<th>Resolution</th>
<th>Image Format</th>
<th>Memory</th>
<th>widthStride &amp; heightStride</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPC input</td>
<td>● Non-8K resizing The following enum values of acldevppPixelFormat are supported: PIXEL_FORMAT_YUV_400 = 0, // 0, YUV400 8bit PIXEL_FORMAT_YUV_SEM IPLANAR_420 = 1, // 1, YUV420SP NV12 8bit PIXEL_FORMAT_YUV_SEM IPLANAR_422 = 2, // 2, YUV422SP NV12 8bit PIXEL_FORMAT_YUV_SEM IPLANAR_444 = 3, // 3, YUV444SP NV12 8bit PIXEL_FORMAT_YUV_SEM IPLANAR_422P = 4, // 4, YUV422SP NV21 8bit PIXEL_FORMAT_YUV_SEM IPLANAR_444P = 5, // 5, YUV444SP NV21 8bit PIXEL_FORMAT_YUV_PAP CKE_422 = 7, // 7, YUV422P YUYV 8bit PIXEL_FORMAT_YUV_PAP CKE_422P = 8, // 8, YUV422P UYVY 8bit PIXEL_FORMAT_YUV_PAP CKE_444 = 9, // 9, YUV444P YYUY 8bit PIXEL_FORMAT_YUV_PAP CKE_422P = 10, // 10, YUV444P VYUY 8bit PIXEL_FORMAT_YUV_PAP CKE_444P = 11, // 11, YUV444P 8bit PIXEL_FORMAT_RGB_888 = 12, // 12, RGB888 PIXEL_FORMAT_BGR_888 = 13, // 13, BGR888 PIXEL_FORMAT_ARGB_88 88 = 14, // 14, ARGB8888 PIXEL_FORMAT_ABGR_88 88 = 15, // 15, ABGR8888 PIXEL_FORMAT_RGBA_88 88 = 16, // 16, RGBA8888 PIXEL_FORMAT_BGRA_88 88 = 17, // 17, BGRA8888 PIXEL_FORMAT_YUV_SEM I PLANNER_420_10BIT = 18, // 18, YUV420SP 10bit PIXEL_FORMAT_YUV_SEM I PLANNER_422_10BIT = 19, // 19, YUV422SP 10bit PIXEL_FORMAT_YUV_SEM I PLANNER_422P_10BIT = 20, // 20, YUV422SP 10bit PIXEL_FORMAT_YUV_PAP CKE_422_10BIT = 21, // 21, YUV422P 10bit PIXEL_FORMAT_YUV_PAP CKE_444P_10BIT = 22, // 22, YUV444P 10bit</td>
<td>● Calculate widthStride according to the image format. – YUV40 0 SP and YUV42 0 SP: widthStride x heightStride x 3/2 – YUV42 2 SP: widthStride x heightStride x 2 – YUV44 4 SP: widthStride x heightStride x 3 – YUV42 2 Packed: widthStride x heightStride</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VP C Input/Output</td>
<td>Resolution</td>
<td>Image Format</td>
<td>Memory</td>
<td>widthStride &amp; heightStride</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------</td>
<td>--------------</td>
<td>--------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>heightStride ∈ [6, 4096]</td>
<td>i.PLANNER_420_10BIT = 19, // 19, YVU420sp 10bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- XRGB8 888: widthStride/4 ∈ [32, 4096] and heightStride ∈ [6, 4096]</td>
<td>height Stride</td>
<td>YUV44 4 Packed and RGB88 8: widthStride x height Stride</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- 8K resizing: widthStride ∈ (4096, 8192) or heightStride ∈ (4096, 8192)</td>
<td>Allocate device memory by calling acldvppMalloc and free device memory by calling acldvppFree.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- XRGB8 888: widthStride x height Stride</td>
<td>occupies 2 bytes).</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- YUV44 4 Packed and RGB88 8: Round up the input width to the nearest multiple of 16 and multiply the result by 3 (that is, each pixel occupies 3 bytes).</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

 Allocate device memory by calling acldvppMalloc and free device memory by calling acldvppFree. |

- XRGB8 888: Round up the input width to the nearest multiple of 16 and multiply the result by 4 (that
<table>
<thead>
<tr>
<th>VP C Input/Output</th>
<th>Resolution</th>
<th>Image Format</th>
<th>Memory</th>
<th>widthStride &amp; heightStride</th>
</tr>
</thead>
</table>

is, each pixel occupies 4 bytes).

- **heightStride**: Round up the input height to the nearest multiple of 2.
<table>
<thead>
<tr>
<th>VP C Input/Output</th>
<th>Resolution</th>
<th>Image Format</th>
<th>Memory</th>
<th>widthStride &amp; heightStride</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPC output</td>
<td>32 x 6 to 4096 x 4096</td>
<td>The following enum values of aclDvppPixelFormat are supported:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // 1, YUV420SP NV12 8bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PIXEL_FORMAT_YVU_SEMIPLANAR_420 = 2, // 2, YUV420SP NV21 8bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The start address must be 16-byte aligned.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Calculate the output memory size according to the image format.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- YUV400 SP and YUV420 SP: widthStride x heightStride x 3/2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- YUV422 SP: widthStride x heightStride x 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- YUV444 4 SP: widthStride x heightStride x 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- YUV422 Packed: widthStride x</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>● widthStride: Round up the input width to the nearest multiple of 16.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>● heightStride: Round up the input height to the nearest multiple of 2.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VP C Input/Output</td>
<td>Resolution</td>
<td>Image Format</td>
<td>Memory</td>
<td>widthStride &amp; heightStride</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------</td>
<td>--------------</td>
<td>--------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>height Stride</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>YUV44 4 Packed and RGB888 8: widthStride \times height Stride</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>XRGB888: widthStride \times height Stride</td>
</tr>
</tbody>
</table>

Allocate device memory by calling `acldvppMalloc` and free device memory by calling `acldvppFree`.

- The resizing ratio of the width or height must be within the range of \([1/32, 16]\).
- For `pasteArea`, the left-offset must be rounded up to the nearest multiple of 16.
VPC Function Diagram and Important Terms

**Figure 7-2** VPC function diagram (cropping+resizing+pasting)

![Figure 7-2 VPC function diagram (cropping+resizing+pasting)](image)

**Figure 7-3** VPC function diagram (stitching)

![Figure 7-3 VPC function diagram (stitching)](image)

**Table 7-3** Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>widthStride</td>
<td>Space occupied by each line of pixels in the buffer. <strong>widthStride</strong> of an RGB image is calculated differently from that of a YUV image. For details about the restrictions, see <strong>Table 7-2</strong>.</td>
</tr>
<tr>
<td>heightStride</td>
<td>Number of pixel lines in the buffer. For details about the restrictions, see <strong>Table 7-2</strong>.</td>
</tr>
</tbody>
</table>
Term | Definition
--- | ---
Top/ Bottom/ Left/ Right offset | Configure the top offset, bottom offset, left offset, and right offset for the following purposes: (1) Specify the position of the cropArea or pasteArea. (2) Specify the width and height of the cropArea or pasteArea by using the following formulas: Right offset – Left offset + 1 = Width; Bottom offset – Top offset + 1 = Height.
- Left offset: horizontal offset of points 1 and 3 in the cropArea or pasteArea relative to point 0
- Right offset: horizontal offset of points 2 and 4 in the cropArea or pasteArea relative to point 0
- Top offset: vertical offset of points 1 and 2 in the cropArea or pasteArea relative to point 0
- Bottom offset: vertical offset of points 3 and 4 in the cropArea or pasteArea relative to point 0

cropArea | Cropping area
The minimum resolution is 10 x 6, and the maximum resolution is 4096 x 4096.
pasteArea | Pasting area in the canvas. The minimum resolution is 10 x 6, and the maximum resolution is 4096 x 4096.
Restrictions:
- For the pasteArea, the left offset and the top offset must be even numbers, and the right offset and the bottom offset must be odd numbers.
- The cropArea cannot be larger than the input image, and the pasteArea cannot be larger than the canvas.
- The pasteArea can be directly mapped on the leftmost side of the canvas, that is, the left offset of the output image is 0.
- Up to 256 pasteAreas are allowed.
- Round up the left offset of pasteArea relative to the output image to the nearest multiple of 16.
- Round up the output pasteArea width to the nearest multiple of 16. Otherwise, padding is performed to meet the alignment requirement.

**Performance Specifications**

- **For non-8K resizing**, the performance specifications in basic scenarios are as follows:
  For 1080p images, if image data is copied from the host to the device, the maximum total frame rate is about 1000 fps due to the limited copy bandwidth.
  For 4K images, if the image data is copied from the host to the device, the maximum total frame rate is about 250 fps due to the copy bandwidth limit.
### Scenario | Total Frame Rate
---|---
- Input resolution: 1080p (1920 x 1080)  
- Output resolution: 1080p (1920 x 1080)  
- Input/Output format: YUV420 SP  
- \(n\) channels (\(n < 4\), one channel corresponds to one thread) | \(n \times 360\)fps
- Input resolution: 1080p (1920 x 1080)  
- Output resolution: 1080p (1920 x 1080)  
- Input/Output format: YUV420 SP  
- \(n\) channels (\(n \geq 4\), one channel corresponds to one thread) | 1440fps
- Input resolution: 4K (3840 x 2160)  
- Output resolution: 4K (3840 x 2160)  
- Input/Output format: YUV420 SP  
- \(n\) channels (\(n < 4\), one channel corresponds to one thread) | \(n \times 90\)fps
- Input resolution: 4K (3840 x 2160)  
- Output resolution: 4K (3840 x 2160)  
- Input/Output format: YUV420 SP  
- \(n\) channels (\(n \geq 4\), one channel corresponds to one thread) | 360fps

- For **8K resizing**, the VPC performance is closely related to the output resolution. A higher output resolution indicates longer processing time and lower performance. The performance specifications in typical scenarios are as follows:

### Scenario | Total Frame Rate
---|---
- Input resolution: 8K (7680 x 4320)  
- Output resolution: 1080p (1920 x 1080)  
- Input/Output format: YUV420 SP  
- \(n\) channels (\(n < 4\), one channel corresponds to one thread) | \(n \times 4\)fps
<table>
<thead>
<tr>
<th>Scenario</th>
<th>Total Frame Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Input resolution: 8K (7680 x 4320)</td>
<td>16fps</td>
</tr>
<tr>
<td>● Output resolution: 1080p (1920 x 1080)</td>
<td></td>
</tr>
<tr>
<td>● Input/Output format: YUV420 SP</td>
<td></td>
</tr>
<tr>
<td>● ( n ) channels (( n \geq 4 ), one channel corresponds to one thread)</td>
<td></td>
</tr>
<tr>
<td>● Input resolution: 8K (7680 x 4320)</td>
<td>( n \times 1 )fps</td>
</tr>
<tr>
<td>● Output resolution: 4K (3840 x 2160)</td>
<td></td>
</tr>
<tr>
<td>● Input/Output format: YUV420 SP</td>
<td></td>
</tr>
<tr>
<td>● ( n ) channels (( n &lt; 4 ), one channel corresponds to one thread)</td>
<td></td>
</tr>
<tr>
<td>● Input resolution: 8K (7680 x 4320)</td>
<td>4fps</td>
</tr>
<tr>
<td>● Output resolution: 4K (3840 x 2160)</td>
<td></td>
</tr>
<tr>
<td>● Input/Output format: YUV420 SP</td>
<td></td>
</tr>
<tr>
<td>● ( n ) channels (( n \geq 4 ), one channel corresponds to one thread)</td>
<td></td>
</tr>
</tbody>
</table>

### 7.12.4.2 aclDvppVpcResizeAsync

**Description**

Resizes the input image to the output image size. It is an asynchronous interface.

**Prototype**

```c
aclError aclDvppVpcResizeAsync(aclDvppChannelDesc *channelDesc,
                              aclDvppPicDesc *inputDesc,
                              aclDvppPicDesc *outputDesc,
                              aclDvppResizeConfig *resizeConfig,
                              aclrtStream stream)
```
## Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelDesc</td>
<td>Input</td>
<td>Channel description, which must be the same as the value of channelDesc specified in the acldvppCreateChannel call.</td>
</tr>
<tr>
<td>inputDesc</td>
<td>Input</td>
<td>Input image description</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Create image description by calling acldvppCreatePicDesc, and set image description parameters (such as the image format, width, and height) by calling acldvppSetPicDesc.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- For details about the input image resolution, image format, widthStride alignment requirements, and heightStride alignment requirements, see Restrictions.</td>
</tr>
<tr>
<td>outputDesc</td>
<td>Input/Output</td>
<td>Output image description</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Create image description by calling acldvppCreatePicDesc.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- For details about the output image resolution, image format, widthStride alignment requirements, and heightStride alignment requirements, see Restrictions.</td>
</tr>
<tr>
<td>resizeConfig</td>
<td>Input</td>
<td>Image resize configuration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Create image resize configuration by calling acldvppCreateResize-Config.</td>
</tr>
<tr>
<td>stream</td>
<td>Input</td>
<td>Stream</td>
</tr>
</tbody>
</table>

## Returns

0 indicates the success of task delivery. Other values indicate the failure of task delivery.

### 7.12.4.3 acldvppVpcCropAsync

#### Description

Crops the input image and stores the cropped image to the output canvas. It is an asynchronous interface.
The cropped image will be resized again if `cropArea` is different from that of the canvas.

**Prototype**

```c
aclError acldvppVpcCropAsync(acldvppChannelDesc *channelDesc,
                              acldvppPicDesc *inputDesc,
                              acldvppPicDesc *outputDesc,
                              acldvppRoiConfig *cropArea,
                              aclrtStream stream)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelDesc</td>
<td>Input</td>
<td>Channel description, which must be the same as the <code>channelDesc</code> argument passed to the <code>acldvppCreateChannel</code> call.</td>
</tr>
<tr>
<td>inputDesc</td>
<td>Input</td>
<td>Input image description</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Create image description by calling <code>acldvppCreatePicDesc</code>, and set image description parameters (such as the image format, width, and height) by calling <code>acldvppSetPicDesc</code>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● For details about the input image resolution, image format, <code>widthStride</code> alignment requirements, and <code>heightStride</code> alignment requirements, see <code>Restrictions</code>.</td>
</tr>
<tr>
<td>outputDesc</td>
<td>Input/Output</td>
<td>Output image description</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Create image description by calling <code>acldvppCreatePicDesc</code>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● For details about the output image resolution, image format, <code>widthStride</code> alignment requirements, and <code>heightStride</code> alignment requirements, see <code>Restrictions</code>.</td>
</tr>
<tr>
<td>cropArea</td>
<td>Input</td>
<td>Cropping area</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Create area position data by calling <code>acldvppCreateRoiConfig</code>.</td>
</tr>
<tr>
<td>stream</td>
<td>Input</td>
<td>Stream</td>
</tr>
</tbody>
</table>
Returns

0 on successful task delivery; else, failure

7.12.4.4 aclDvppVpcBatchCropAsync

Description

Crops one or more input images and stores the cropped one or more areas to the output canvases. It is an asynchronous interface. A cropped area will be resized again if the cropArea is different from that of the corresponding canvas. It is an asynchronous interface.

Prototype

```c
aclError aclDvppVpcBatchCropAsync(acldvppChannelDesc *channelDesc,
                                   acldvppBatchPicDesc *srcBatchPicDescs,
                                   uint32_t *roiNums,
                                   uint32_t size,
                                   acldvppBatchPicDesc *dstBatchPicDescs,
                                   aclDvppRoiConfig *cropAreas[],
                                   aclrtStream stream)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelDesc</td>
<td>Input</td>
<td>Channel description, which must be the same as the channelDesc argument passed to the aclDvppCreateChannel call.</td>
</tr>
<tr>
<td>srcBatchPicDescs</td>
<td>Input</td>
<td>Description of the input images Call aclDvppCreateBatchPicDesc to create the image description.</td>
</tr>
<tr>
<td>roiNums</td>
<td>Input</td>
<td>Number of cropped images, up to 256 roINums is an array of size up to 256. Has the same size value as batchSize in the dstBatchPicDescs struct. roiNums[0]+...+roiNums[size-1] &lt;= 256</td>
</tr>
<tr>
<td>size</td>
<td>Input</td>
<td>Number of elements in the roiNums array, up to 256</td>
</tr>
<tr>
<td>Parameter</td>
<td>Input/Output</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>dstBatchPicDescs</td>
<td>Input/Output</td>
<td>Description of the output images Call <code>acldvppCreateBatchPicDesc</code> to create the image description.</td>
</tr>
<tr>
<td>cropArea</td>
<td>Input</td>
<td>An array for the position of the cropping area Create area data by calling <code>acldvppCreateRoiConfig</code>.</td>
</tr>
<tr>
<td>stream</td>
<td>Input</td>
<td>Stream</td>
</tr>
</tbody>
</table>

**Returns**

0 on successful task delivery; else, failure

### 7.12.4.5 acldvppVpcCropAndPasteAsync

**Description**

Crops the input image and paste the cropped image to the output canvas. It is an asynchronous interface.

The image will be resized again if the width and height of `cropArea` are different from that of `pasteArea`.

**Prototype**

```c
aclError acldvppVpcCropAndPasteAsync(acldvppChannelDesc *channelDesc,
                                   acldvppPicDesc *inputDesc,
                                   acldvppPicDesc *outputDesc,
                                   acldvppRoiConfig *cropArea,
                                   acldvppRoiConfig *pasteArea,
                                   aclrtStream stream)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelDesc</td>
<td>Input</td>
<td>Channel description, which must be the same as the value of <code>channelDesc</code> specified in the <code>acldvppCreateChannel</code> call.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Input/Output</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| inputDesc    | Input        | Input image description  
  - Create image description by calling `acldvppCreatePicDesc`, and set image description parameters (such as the image format, width, and height) by calling `acldvppSetPicDesc`.  
  - For details about the input image resolution, image format, `widthStride` alignment requirements, and `heightStride` alignment requirements, see `Restrictions`. |
| outputDesc   | Input/Output | Output image description  
  - Create image description by calling `acldvppCreatePicDesc`.  
  - For details about the output image resolution, image format, `widthStride` alignment requirements, and `heightStride` alignment requirements, see `Restrictions`. |
| cropArea     | Input        | Cropping area  
  - Create area data by calling `acldvppCreateRoiConfig`. |
| pasteArea    | Input        | Pasting area  
  - Create area data by calling `acldvppCreateRoiConfig`.  
  - The left offset of `pasteArea` must be 16-pixel aligned. |
| stream       | Input        | Stream |

**Returns**

0 on successful task delivery; else, failure

**7.12.4.6 acldvppVpcBatchCropAndPasteAsync**

**Description**

Crops the input images and pastes the cropped areas to the output canvases. It is an asynchronous interface.

The image will be resized again if the width and height of `cropArea` are different from that of `pasteArea`. 
Prototype

```c
aclError aclVppVpcBatchCropAndPasteAsync (acldvppChannelDesc *channelDesc,
acldvppBatchPicDesc *srcBatchPicDesc,
uint32_t *roiNums,
uint32_t size,
acldvppBatchPicDesc *dstBatchPicDesc,
acldvppRoiConfig *cropAreas[],
acldvppRoiConfig *pasteAreas[],
aclrtStream stream)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelDesc</td>
<td>Input</td>
<td>Channel description, which must be the same as the channelDesc argument passed to the acldvppCreateChannel call.</td>
</tr>
<tr>
<td>srcBatchPicDescs</td>
<td>Input</td>
<td>Description of the input images Call acldvppCreateBatchPicDesc to create the image description.</td>
</tr>
<tr>
<td>roiNums</td>
<td>Input</td>
<td>Number of cropping areas, up to 256 roiNums is an array of size up to 256. Has the same size value as batchSize in the dstBatchPicDescs struct. roiNums[0]+...+roiNums[size-1] &lt;= 256</td>
</tr>
<tr>
<td>size</td>
<td>Input</td>
<td>Number of elements in the roiNums array, up to 256</td>
</tr>
<tr>
<td>dstBatchPicDescs</td>
<td>Input/Output</td>
<td>Description of the output images Call acldvppCreateBatchPicDesc to create the image description.</td>
</tr>
</tbody>
</table>
### Parameter | Input/Output | Description
--- | --- | ---
**cropArea** | Input | An array for the position of the cropping area. Create area data by calling `acldvppCreateRoiConfig`. The `cropAreas` or `pasteAreas` array size is the same as the value of `batchSize` in the `dstBatchPicDescs` struct.

**pasteArea** | Input | Pasting area
- Create area data by calling `acldvppCreateRoiConfig`.
- The left offset of `pasteArea` must be 16-pixel aligned.
- The `cropAreas` or `pasteAreas` array size is the same as the value of `batchSize` in the `dstBatchPicDescs` struct.

**stream** | Input | Stream

### Returns
0 on successful task delivery; else, failure

### 7.12.4.7 acldvppVpcConvertColorAsync

**Description**
Performs CSC on the input image. The resolution of the output image must be the same as that of the input image. It is an asynchronous interface. The current version does not support this interface.

**Prototype**

```c
aclError acldvppVpcConvertColorAsync(acldvppChannelDesc *channelDesc,
                                      acldvppPicDesc *inputDesc,
                                      acldvppPicDesc *outputDesc,
                                      aclrtStream stream)
```
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelDesc</td>
<td>Input</td>
<td>Channel description, which must be the same as the <code>channelDesc</code> argument passed to the <code>acldvppCreateChannel</code> call.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Input/Output</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| inputDesc   | Input        | **Input image description**<br>- Create image description by calling `acldvppCreatePicDesc`, and set image description parameters (such as the image format, width, and height) by calling `acldvppSetPicDesc`.<br>- For details about the input image resolution, `widthStride` and `heightStride` alignment requirements, see **Restrictions**.<br>- The supported input image formats are as follows:<br>  PIXEL_FORMAT_YUV_400, // When the input image format is YUV400, the output format must be YUV400.<br>  PIXEL_FORMAT_YUV_SEMIPLAN_AR_420,<br>  PIXEL_FORMAT_YUV_SEMIPLAN_AR_420,<br>  PIXEL_FORMAT_YUV_SEMIPLAN_AR_422,<br>  PIXEL_FORMAT_YUV_SEMIPLAN_AR_422,<br>  PIXEL_FORMAT_YUV_SEMIPLAN_AR_444,<br>  PIXEL_FORMAT_YUV_SEMIPLAN_AR_444,<br>  PIXEL_FORMAT_YUYV_PACKED_4_22,<br>  PIXEL_FORMAT_UYVY_PACKED_4_22,<br>  PIXEL_FORMAT_YVYU_PACKED_4_22,<br>  PIXEL_FORMAT_VYUY_PACKED_4_22,<br>  PIXEL_FORMAT_YUVPACKED_4_44,<br>  PIXEL_FORMAT_RGB_888,<br>  PIXEL_FORMAT_BGR_888,<br>  PIXEL_FORMAT_ARGB_8888,<br>  PIXEL_FORMAT_ABGR_8888,<br>  PIXEL_FORMAT_RGBA_8888,
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
</table>
|           |              | **PIXEL_FORMAT_BGRA_8888,**  
|           |              | **PIXEL_FORMAT_YUV_SEMI_PLAN**  
|           |              | **NER_420_10BIT,**  
|           |              | **PIXEL_FORMAT_YVU_SEMI_PLAN**  
|           |              | **NER_420_10BIT** |
| outputDesc | Input/Output | Output image description  
|           |              | - Create image description by calling **acldvppCreatePicDesc**.  
|           |              | - For details about the output image resolution, **widthStride** and **heightStride** alignment requirements, see **Restrictions**.  
|           |              | - The supported output image formats are as follows:  
|           |              | **PIXEL_FORMAT_YUV_400,**  
|           |              | **PIXEL_FORMAT_YVU_SEMIPLAN AR_420,**  
|           |              | **PIXEL_FORMAT_YVU_SEMIPLAN AR_420,**  
|           |              | **PIXEL_FORMAT_YVU_SEMIPLAN AR_422,**  
|           |              | **PIXEL_FORMAT_YVU_SEMIPLAN AR_422,**  
|           |              | **PIXEL_FORMAT_YVU_SEMIPLAN AR_444,**  
|           |              | **PIXEL_FORMAT_RGB_888,**  
|           |              | **PIXEL_FORMAT_BGR_888,**  
|           |              | **PIXEL_FORMAT_ARGB_8888,**  
|           |              | **PIXEL_FORMAT_ABGR_8888,**  
|           |              | **PIXEL_FORMAT_RGBA_8888,**  
|           |              | **PIXEL_FORMAT_BGRA_8888** |
| stream    | Input        | Stream |

**Returns**

0 indicates the success of task delivery. Other values indicate the failure of task delivery.
7.12.4.8 acldvppVpcPyrDownAsync

Description
Performs pyramid resizing to resize the input image to half the original size. It is an asynchronous interface. The current version does not support this interface.

Prototype

```c
aclError acldvppVpcPyrDownAsync(acldvppChannelDesc *channelDesc,
                                 acldvppPicDesc *inputDesc,
                                 acldvppPicDesc *outputDesc,
                                 void *reserve,
                                 aclrtStream stream)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelDesc</td>
<td>Input</td>
<td>Channel description, which must be the same as the channelDesc argument passed to the acldvppCreateChannel call.</td>
</tr>
</tbody>
</table>
| inputDesc      | Input        | Input image description
  - Create image description by calling acldvppCreatePicDesc, and set image description parameters (such as the image format, width, and height) by calling acldvppSetPicDesc.
  - Resolution: 20 x 12 to 2048 x 2048
  - Format: PIXEL_FORMAT_YUV_400
  - widthStride alignment: 16-pixel alignment
  - heightStride alignment: no requirement |
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
</table>
| outputDesc | Input/Output | Output image description  
  Create image description by calling `acldvppCreatePicDesc`.  
  Resolution: input image resolution/2  
  Format: `PIXEL_FORMAT_YUV_400`  
  `widthStride` alignment: 16-pixel alignment  
  `heightStride` alignment: no requirement |
| reserve    | Input        | Reserved    |
| stream     | Input        | Stream      |

**Returns**

0 indicates the success of task delivery. Other values indicate the failure of task delivery.

### 7.12.5 JPEGD

#### 7.12.5.1 Functions and Restrictions

**Functions and Restrictions**

The JPEG decoder (JPEGD) decodes .jpg, .jpeg, JPG, and .JPEG images. For the image decoding involving formats not supported by the hardware, software decoding is used.

- About input
  - Input resolution range:
    - 32 x 32 to 8192 x 8192
  - Input formats:
    - Only Huffman encoding is supported. The color space of the stream is YUV, and the subsample of the stream is 444, 422, 420, or 400.
    - Arithmetic coding is not supported.
    - The progressive JPEG format is not supported.
    - The JPEG2000 format is not supported.
  - Input memory:
- The size of the input memory is the size of the input image.
- The start address of the input memory must be 128-byte aligned. Allocate device memory by calling `acldvppMalloc` and free device memory by calling `acldvppFree`.

- About output
  - Output formats:
    After decoding for different source images, the JPEGD outputs images are in the following formats:
    JPEG (YUV444 SP) -> YUV444 SP with V component before U component, YUV420 SP with V component before U component, or YUV420 SP with U component before V component
    JPEG (YUV422 SP) -> YUV422 SP with V component before U component, YUV420 SP with V component before U component, or YUV420 SP with U component before V component
    JPEG (YUV420 SP) -> YUV420 SP with V component before U component, or YUV420 SP with U component before V component
    JPEG (YUV400) -> YUV420 SP with U and V components padded with 0x80
  - Output memory:
    - Calculate the output memory size according to the image format.
      YUV420 SP: \( \text{widthStride} \times \text{heightStride} \times 3/2 \)
      YUV422 SP: \( \text{widthStride} \times \text{heightStride} \times 2 \)
      YUV444 SP: \( \text{widthStride} \times \text{heightStride} \times 3 \)
    - The start address of the output memory must be 128-byte aligned. Allocate device memory by calling `acldvppMalloc` and free device memory by calling `acldvppFree`. For huge memory: Memory size = Output memory size + \((n - 1) \times \text{AlignTo128}(\text{Output memory size} + 8)\), where, \(n\) indicates the number of images
  - Output width and height:
    - \(\text{widthStride}\): Round up the input width to the nearest multiple of 128.
    - \(\text{heightStride}\): Round up the input height to the nearest multiple of 16.

- Hardware restrictions:
  - A maximum of four Huffman tables are supported, including two direct coefficient (DC) tables and two alternating coefficient (AC) tables.
  - A maximum of three quantization tables are supported.
  - Only 8-bit sampling is supported.
  - Only sequentially-encoded images can be decoded.
  - Only JPEG decoding based on discrete cosine transform (DCT) is supported.
  - Only one start of scan (SOS) flag is supported for image decoding.
Software restrictions:
- A maximum of three SOS flags are supported for image decoding.
- Abnormal image decoding with insufficient minimum coded unit (MCU) data is supported.

Performance Specifications

The performance specifications of the JPEGD are based on the hardware decoding performance. The JPEGD hardware decoding does not support the image decoding with three SOS flags. For image formats not supported by the hardware, software decoding is used. The software decoding performance for reference is 1080p x 1 channel at 15 fps.

1080p indicates a resolution of 1920 x 1080 pixels. 4K indicates a resolution of 3840 x 2160 pixels.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Total Frame Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-channel 1080p</td>
<td>128fps</td>
</tr>
<tr>
<td>$n$-channel 1080p ($n \geq 2$)</td>
<td>256fps</td>
</tr>
<tr>
<td>1-channel 4K</td>
<td>32fps</td>
</tr>
<tr>
<td>$n$-channel 4K ($n \geq 2$)</td>
<td>64fps</td>
</tr>
</tbody>
</table>

7.12.5.2 acldvppJpegDecodeAsync

Description

Decodes JPG, JPEG, JPG, and JPEG images. It is an asynchronous interface.

Prototype

```c
aclError acldvppJpegDecodeAsync(acldvppChannelDesc *channelDesc,
                                const void *data,
                                uint32_t size,
                                acldvppPicDesc *outputDesc,
                                aclrtStream stream)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelDesc</td>
<td>Input</td>
<td>Channel description, which must be the same as the <code>channelDesc</code> argument passed to the <code>acldvppCreateChannel</code> call.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Input/Output</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>data</td>
<td>Input</td>
<td>Memory address of the input image</td>
</tr>
<tr>
<td>size</td>
<td>Input</td>
<td>Actual size of the input image, in bytes</td>
</tr>
<tr>
<td>outputDesc</td>
<td>Input/Output</td>
<td>Output image description&lt;br&gt;  ● Create image description by calling <code>acldvppCreatePicDesc</code>.&lt;br&gt;  ● The width and height of the input JPEG image can be obtained by calling <code>acldvppJpegGetImageInfo</code>.&lt;br&gt;  ● The memory size of the output image can be obtained by calling <code>acldvppJpegPredictDecSize</code> in advance.&lt;br&gt;  ● The output image format can be set to the JPEG source image encoding format, NV12 format, or NV21 format.&lt;br&gt;  ● For details about the width and height alignment requirements of the destination image, see 7.12.5.1 Functions and Restrictions.</td>
</tr>
<tr>
<td>stream</td>
<td>Input</td>
<td>Stream</td>
</tr>
</tbody>
</table>

**Returns**

0 indicates the success of task delivery. Other values indicate the failure of task delivery.

**7.12.5.3 acldvppJpegGetImageInfo**

**Description**

Reads the width and height of the JPEG image from the memory on the host that stores the JPEG image data.

**Prototype**

```c
aclError acldvppJpegGetImageInfo(const void *data,
                                  uint32_t size,
                                  uint32_t *width,
                                  uint32_t *height,
```
int32_t *components)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Input</td>
<td>Address of the memory on the host that stores the JPEG image data</td>
</tr>
<tr>
<td>size</td>
<td>Input</td>
<td>Memory size, in bytes</td>
</tr>
<tr>
<td>width</td>
<td>Output</td>
<td>Width of the image</td>
</tr>
<tr>
<td>height</td>
<td>Output</td>
<td>Height of the image</td>
</tr>
<tr>
<td>components</td>
<td>Output</td>
<td>Number of color channels</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.12.5.4 aclDvppJpegPredictDecSize

Description

Predicts the output memory size required for the decoded JPEG image based on the memory used for storing JPEG image data on the host.

Prototype

`aclError aclDvppJpegPredictDecSize(const void *data, uint32_t dataSize, aclDvppPixelFormat outputPixelFormat, uint32_t *decSize)`

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Input</td>
<td>Address of the memory on the host that stores the JPEG image data</td>
</tr>
<tr>
<td>size</td>
<td>Input</td>
<td>Memory size, in bytes</td>
</tr>
<tr>
<td>outputPixelFormat</td>
<td>Input</td>
<td>Format of the decoded output image</td>
</tr>
<tr>
<td>decSize</td>
<td>Output</td>
<td>Size of the output memory required for the decoded JPEG images, in bytes</td>
</tr>
</tbody>
</table>
Returns

0 indicates success. Other values indicate failure.

7.12.6 JPEGE

7.12.6.1 Functions and Restrictions

Functions and Restrictions

The JPEG encoder (JPEGE) encodes YUV images into JPEG image files, for example, .jpg.

- About input
  - Input resolution range: 32 x 32 to 8192 x 8192
  - Input image formats:
    - YUV422 Packed (YUYV, YVYU, UYVY, VYUY)
    - YUV420 SP (NV12 and NV21)
  - Input width and height:
    - Round up widthStride to the nearest multiple of 16, for example, 128. For YUV422 Packed data, round up widthStride to the nearest multiple of 16, greater than twice the input width.
    - Round up heightStride to the input height or the nearest multiple of 16.
  - Input memory:
    - Calculate the input memory size according to the image format.
      YUV422 Packed: widthStride x heightStride
      YUV420 SP: widthStride x heightStride x 3/2
    - The start address of the input memory must be 128-byte aligned. Allocate device memory by calling acldvppMalloc and free device memory by calling acldvppFree.

- About output
  - The JPEG outputs image files in JPEG format, for example, .jpg.
  - Only Huffman encoding is supported. Arithmetic encoding and progressive encoding are not supported.
  - Output memory:
    - The size of the output memory is the size of the encoded image.
    - The start address of the output memory must be 128-byte aligned. Allocate device memory by calling acldvppMalloc and free device memory by calling acldvppFree.
Performance Specifications

1080p indicates a resolution of 1920 x 1080 pixels. 4K indicates a resolution of 3840 x 2160 pixels.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Total Frame Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>n-channel 1080p (n ≥ 1)</td>
<td>64fps</td>
</tr>
<tr>
<td>n-channel 4K (n ≥ 1)</td>
<td>16fps</td>
</tr>
</tbody>
</table>

7.12.6.2 aclDvppJpegEncodeAsync

Description

Encodes YUV images into JPG images. It is an asynchronous interface.

Prototype

```c
aclError aclDvppJpegEncodeAsync(acldvppChannelDesc *channelDesc,
                                acldvppPicDesc *inputDesc,
                                const void *data,
                                uint32_t *size,
                                acldvppJpegeConfig *config,
                                aclrtStream stream);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelDesc</td>
<td>Input</td>
<td>Channel description, which must be the same as the value of channelDesc specified in the acldvppCreateChannel call.</td>
</tr>
<tr>
<td>inputDesc</td>
<td>Input</td>
<td>Input image description</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Create image description by calling acldvppCreatePicDesc, and set image description parameters (such as the image format, width, and height) by calling acldvppSetPicDesc.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- For details about the input resolution and format restrictions, see 7.12.6.1 Functions and Restrictions.</td>
</tr>
</tbody>
</table>
### Table

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Input</td>
<td>Output memory address for storing encoded data</td>
</tr>
<tr>
<td>size</td>
<td>Input/Output</td>
<td>Output memory size, in bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When <code>size</code> is used as an input, you can estimate the output memory size by calling <code>acldvppJpegPredictEncSize</code> in advance. When <code>size</code> is used as an output, it indicates the actual output memory size, which may be different from the memory size estimated by calling <code>acldvppJpegPredictEncSize</code>. The actual output memory size should apply.</td>
</tr>
<tr>
<td>config</td>
<td>Input</td>
<td>Image encoding configuration data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Create the image encoding configuration data by calling <code>acldvppCreateJpegeConfig</code>, and set the encoding configuration data by calling <code>acldvppSetJpegeConfigLevel</code>.</td>
</tr>
<tr>
<td>stream</td>
<td>Input</td>
<td>Stream</td>
</tr>
</tbody>
</table>

### Returns

0 indicates the success of task delivery. Other values indicate the failure of task delivery.

### 7.12.6.3 acldvppJpegPredictEncSize

#### Description

Predicts the output memory size required for the encoded image based on the input image description and image encoding configuration data.

#### Prototype

```c
aclError acldvppJpegPredictEncSize(const acldvppPicDesc *inputDesc, const acldvppJpegeConfig *config, uint32_t *size)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputDesc</td>
<td>Input</td>
<td>Input image description. Create image description by calling acldvppCreatePicDesc, and set image description parameters (such as the image format, width, and height) by calling acldvppSetPicDesc.</td>
</tr>
<tr>
<td>config</td>
<td>Input</td>
<td>Image encoding configuration data. Create image encoding configuration data by calling acldvppCreateJpegeConfig.</td>
</tr>
<tr>
<td>size</td>
<td>Output</td>
<td>Predicted size of the output memory required for the encoded JPEG image, in bytes. The predicted output memory is greater than the actual output memory.</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.12.7 VDEC

7.12.7.1 Functions and Restrictions

The video decoder (VDEC) decodes videos. After VPC processing, the VDEC outputs YUV420 SP images (including NV12 and NV21).

- About input
  - Input resolution range: 128 x 128 to 4096 x 4096
  - Input formats:
    - H.264 BP/MP/HP Level 5.1 YUV420 encoded streams
    - H.265 8-bit/10-bit Level 5.1 YUV420 encoded streams
  - Input memory:
    - Allocate device memory by calling aclrtMalloc and free memory by calling aclrtFree. Alternatively, allocate device memory by calling acldvppMalloc and free memory by calling acldvppFree.

- About output
  - Output image formats:
- YUV420SP NV12
- YUV420SP NV21
- Output memory:
  - Calculate the output memory size according to the image format.
  - YUV420 SP: $\text{widthStride} \times \text{heightStride} \times 3/2$
  - The start address of the output memory must be 16-byte aligned.
  - Allocate device memory by calling `aclrtMalloc` and free device memory by calling `aclrtFree`.
- Output width and height:
  - $\text{widthStride}$: Round up the input width to the nearest multiple of 16.
  - $\text{heightStride}$: Round up the input height to the nearest multiple of 2.
- Bad frames or frame loss in the streams may cause VDEC frame loss.
- The VDEC cannot decode the streams encoded in interlaced scanning mode.

### 7.12.7.2 aclvdecSendFrame

#### Description
Sends an image (as well as the output memory) to the decoder for decoding. It is an asynchronous interface.

`aclrtLaunchCallback` is encapsulated in the `aclvdecSendFrame` interface to add a callback function that needs to be executed on host to the stream task queue. When implementing the VDEC function, `aclrtLaunchCallback` does not need to be called separately.

#### Restrictions
- Before sending data, ensure that a channel is created. Otherwise, an error code is returned.
- Streams must be sent in the unit of frames. Only one complete frame of streams is sent at a time.
- Empty stream packet with end of stream (EOS) = 0 (stream length is 0 or stream address is empty) cannot be sent.
- After all streams are sent, an empty stream packet with EOS = 1 can be sent, indicating the end of the current stream file.
- The recommended number of channels ($n$) for VDEC decoding is as follows:
  - 720p indicates a resolution of 1280 x 720 pixels.
  - 1080p indicates a resolution of 1920 x 1080 pixels.
  - 4K indicates a resolution of 3840 x 2160 pixels.
### Resolution and Frame Rate Specifications

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Total Frame Rate</th>
<th>Single-Channel Frame Rate (n channels)</th>
<th>Memory Consumption Per VDEC Channel (for Reference Only)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3840*2160</td>
<td>120fps</td>
<td>120 fps/n (n = 4) recommended, 30 fps per channel.)</td>
<td>About 170 MB</td>
</tr>
<tr>
<td>1920*1080</td>
<td>480fps</td>
<td>480 fps/n (n = 16) recommended, 30 fps per channel.)</td>
<td>About 86 MB</td>
</tr>
<tr>
<td>≤1280*720</td>
<td>960fps</td>
<td>960 fps/n (n = 32) recommended, 30 fps per channel.)</td>
<td>About 70 MB</td>
</tr>
</tbody>
</table>

- **Suggestions on the Number of Decoding Channels and Frame Rate**

  The specifications in the following table are for reference only. If the number of channels started in a single process exceeds the specifications in the following table, memory or performance insufficiency may occur, leading to decoding channel creation failures and slow decoding.

  720p indicates a resolution of 1280 x 720 pixels. 1080p indicates a resolution of 1920 x 1080 pixels. 4K indicates a resolution of 3840 x 2160 pixels.

<table>
<thead>
<tr>
<th>Typical Resolution</th>
<th>Suggested Channel Count Per Process (Based on the Input Frame Rate)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Input frame rate ≥ 25 fps</td>
</tr>
<tr>
<td>≤720p</td>
<td>20 fps &lt; input frame rate &lt; 25 fps</td>
</tr>
<tr>
<td></td>
<td>15 fps &lt; input frame rate ≤ 20 fps</td>
</tr>
<tr>
<td></td>
<td>10 fps &lt; input frame rate ≤ 15 fps</td>
</tr>
<tr>
<td></td>
<td>Input frame rate ≤ 10 fps</td>
</tr>
<tr>
<td>1080p</td>
<td>32 channels</td>
</tr>
<tr>
<td></td>
<td>32 channels</td>
</tr>
<tr>
<td></td>
<td>32 channels</td>
</tr>
<tr>
<td></td>
<td>32 channels</td>
</tr>
<tr>
<td>4K</td>
<td>4 channels</td>
</tr>
<tr>
<td></td>
<td>4 channels</td>
</tr>
<tr>
<td></td>
<td>Six inputs</td>
</tr>
<tr>
<td></td>
<td>8 channels</td>
</tr>
<tr>
<td></td>
<td>12 channels</td>
</tr>
</tbody>
</table>

### Prototype

```c
aclError aclvdecSendFrame(aclvdecChannelDesc *channelDesc,
    acldvppStreamDesc *input,
    acldvppPicDesc *output,
    aclvdecFrameConfig *config,
    void* userData);
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelDesc</td>
<td>Input</td>
<td>Channel description, which must be the same as the <code>channelDesc</code> argument passed to the <code>aclvdecCreateChannel</code> call. Specify the callback function for video decoding in channel description.</td>
</tr>
<tr>
<td>input</td>
<td>Input</td>
<td>Input stream description. You need to allocate input memory in advance.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Create the stream description by calling <code>acldvppCreateStreamDesc</code>, and set attributes (such as stream format) of the video stream information by calling <code>acldvppSetStreamDesc</code>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● For details about requirements on the resolution and video format of input video streams, see 7.12.7.1 Functions and Restrictions.</td>
</tr>
<tr>
<td>output</td>
<td>Input</td>
<td>Output image description. You need to allocate output memory in advance.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Create image description by calling <code>acldvppCreatePicDesc</code>, and set image description parameters (such as the image format, width, and height) by calling <code>acldvppSetPicDesc</code>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● For details about the requirements on output image format, see 7.12.7.1 Functions and Restrictions.</td>
</tr>
<tr>
<td>config</td>
<td>Input</td>
<td>Reserved parameter for decoding configuration. Currently, pass <code>NULL</code>.</td>
</tr>
<tr>
<td>userdata</td>
<td>Input</td>
<td>User-defined data</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.
7.12.7.3 aclvdecCallback

Description

Video decoding callback function. It is a synchronous interface. This callback function needs to be customized.

Restrictions

- The channel cannot be destroyed in the callback function. Otherwise, a deadlock occurs during application execution.
- The delay for handling the callback function must meet the frame sending rate requirement. Otherwise, the real-time performance of video frame processing of aclvdecSendFrame is affected.
- The deregistration of callback thread needs to be executed only after all callbacks are handled (that is, after the deregistration of the stream sending thread).

Prototype

```c
void (* aclvdecCallback) (acldvppStreamDesc * input, acldvppPicDesc * output, void* userData)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>Input</td>
<td>Description of the input stream, which is the same as the input in aclvdecSendFrame</td>
</tr>
<tr>
<td>output</td>
<td>Input</td>
<td>Description of the output image after VDEC decoding. You need to allocate output memory in advance.</td>
</tr>
<tr>
<td>userData</td>
<td>Input</td>
<td>User-defined data</td>
</tr>
</tbody>
</table>

Returns

None

7.12.8 VENC

7.12.8.1 Functions and Restrictions

Functions and Restrictions

The video encoder (VENC) encodes YUV420 SP NV12/NV21 8-bit image data into H.264/H.265 video streams. Multithreaded processes are not supported.
• About input
  – Input resolution range:
    128 x 128 to 1920 x 1920
  – Input formats:
    YUV420 SP NV12/NV21-8bit
  – Input memory:
    Allocate device memory by calling aclDvppMalloc and free device memory by calling aclDvppFree.

• About output
  – Output formats:
    H264 BP/MP/HP
    H.265 MP (slice streams only)
  – Output memory:
    The output memory is managed by the system.

Performance Specifications

1080p indicates a resolution of 1920 x 1080 pixels.

<table>
<thead>
<tr>
<th>Scenario Example</th>
<th>Total Frame Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$-channel 1080p (one process corresponds to one channel)</td>
<td>30fps</td>
</tr>
</tbody>
</table>

7.12.8.2 aclvencSendFrame

Description

Sends an image to the encoder for encoding. It is an asynchronous interface.

Restrictions

• Before sending data, ensure that a channel is created. Otherwise, an error code is returned.
• Empty stream packet with end of stream (EOS) = 0 (stream length is 0 or stream address is empty) cannot be sent.
• When an empty image with EOS = 1 is sent, the current encoding ends.

Prototype

```c
aclError aclvencSendFrame(aclvencChannelDesc *channelDesc,
                           aclDvppPicDesc *input,
                           void *reserve,
                           aclvencFrameConfig *config,
                           void *userdata)
```
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelDesc</td>
<td>Input</td>
<td>Channel description, which must be the same as the <code>channelDesc</code> argument passed to the <code>aclvencCreateChannel</code> call. Specify the callback function for video encoding in channel description.</td>
</tr>
</tbody>
</table>
| input      | Input        | Input image description. You need to allocate input memory in advance.  
- Create image description by calling `acldvppCreatePicDesc`, and set image description parameters (such as the image format, width, and height) by calling `acldvppSetPicDesc`.  
- For details about the input resolution and format restrictions, see [7.12.8.1 Functions and Restrictions](#). |
| reserve    | Input        | This parameter is reserved. |
| config     | Input        | Single-frame configuration data |
| userdata   | Input        | User-defined data |

### Returns

0 indicates success. Other values indicate failure.

### 7.12.8.3 aclvencCallback

**Description**

Video encoding callback function. It is a synchronous interface. This callback function needs to be customized.

**Restrictions**

- The channel cannot be destroyed in the callback function. Otherwise, a deadlock occurs during application execution.
- The delay for handling the callback function must meet the frame sending rate requirement. Otherwise, the real-time performance of video frame processing of `aclvencSendFrame` is affected.
- Deregister the callback thread after all callbacks are executed.

Prototype

```c
void (*aclvencCallback)(acldvppPicDesc *input, acldvppStreamDesc *output, void *userdata)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>Input</td>
<td>Input image description</td>
</tr>
<tr>
<td>output</td>
<td>Input</td>
<td>Output image description</td>
</tr>
<tr>
<td>userdata</td>
<td>Input</td>
<td>User-defined data</td>
</tr>
</tbody>
</table>

Returns

None

### 7.13 Log Management

#### 7.13.1 aclAppLog

Description

Logs events in log files in `/var/log/npu/slog` on the host.

The `ACL_APP_LOG` macro encapsulates the `aclAppLog` interface. You are advised to pass the variable parameters in the log level, log description, and `fmt` to the `ACL_APP_LOG` macro call.

```c
#define ACL_APP_LOG(level, fmt, ...) \
    aclAppLog(level, __FUNCTION__, __FILE__, __LINE__, fmt, ##__VA_ARGS__)
```

Prototype

```c
void aclAppLog(aclLogLevel logLevel, const char *func, const char *file, uint32_t line, const char *fmt, ...)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logLevel</td>
<td>Input</td>
<td>Log level</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>typedef enum {</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACL_DEBUG = 0,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACL_INFO = 1,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACL_WARNING = 2,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACL_ERROR = 3,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>} aclLogLevel;</td>
</tr>
<tr>
<td>Parameter</td>
<td>Input/Output</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>func</td>
<td>Input</td>
<td>Interface in which the <code>aclAppLog</code> is called. The fixed value is <code>__FUNCTION__</code>.</td>
</tr>
<tr>
<td>file</td>
<td>Input</td>
<td>File in which the <code>aclAppLog</code> is called. The fixed value is <code>__FILE__</code>.</td>
</tr>
<tr>
<td>line</td>
<td>Input</td>
<td>Line in which the <code>aclAppLog</code> is called. The fixed value is <code>__LINE__</code>.</td>
</tr>
<tr>
<td>fmt</td>
<td>Input</td>
<td>Log description. When the formatting function is called, the type and number of parameters in <code>fmt</code> must be the same as that in actuality.</td>
</tr>
<tr>
<td>...</td>
<td>Input</td>
<td>A variable in <code>fmt</code>, added based on the log content</td>
</tr>
</tbody>
</table>

**Returns**

None

**Example**

```
// If fmt contains variable parameters, define them in advance.
uint32_t modelId = 1;
ACL_APP_LOG(ACL_INFO, "load model success, modelId is %u", modelId);
```

### 7.14 Feature Vector Search

In this section, 1:N indicates the ratio of the number of search requests to the number of repositories, and N:M indicates the similarity between two repositories.

Ascend 310 AI Processor does not support APIs described in this section.

#### 7.14.1 aclfvInit

**Description**

Initializes the feature search module.

**Prototype**

```
aclError aclfvInit(uint64_t fsNum)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fsNum</td>
<td>Input</td>
<td>Number of features in the repository, which is used to allocate memory for the feature search module. Value range: [1, 600000000]</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.14.2 aclfvRelease

Description

Deinitializes the feature search module to release the memory space. This API is used in pair with aclfvInit.

Prototype

`aclError aclfvRelease()`

Parameters

None

Returns

0 indicates success. Other values indicate failure.

7.14.3 aclfvRepoAdd

Description

Adds a repository or add features to an existing repository.

Restrictions

- In 1:N adding mode, ensure that the offsets in the aclfvFeatureInfo struct are consecutive for each repository. In N:M adding mode, if the offset is not 0, an error is reported.
- In N:M adding mode, multithreaded adding, search, or deletion is not allowed.

Prototype

`aclError aclfvRepoAdd(aclfvSearchType type, aclfvFeatureInfo *featureInfo, aclrtStream stream)`
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Input</td>
<td>Search type</td>
</tr>
<tr>
<td>featureInfo</td>
<td>Input</td>
<td>Feature description</td>
</tr>
<tr>
<td>stream</td>
<td>Input</td>
<td>Stream</td>
</tr>
</tbody>
</table>

### Returns

0 indicates success. Other values indicate failure.

### 7.14.4 aclfvRepoDel

**Description**

Deletes a repository.

**Restrictions**

In N:M adding mode, multithreaded adding, search, or deletion is not allowed.

**Prototype**

```c
aclError aclfvRepoDel(aclfvSearchType type, aclfvRepoRange *repoRange, aclrtStream stream)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Input</td>
<td>Search type</td>
</tr>
<tr>
<td>repoRange</td>
<td>Input</td>
<td>Repository range</td>
</tr>
<tr>
<td>stream</td>
<td>Input</td>
<td>Stream</td>
</tr>
</tbody>
</table>

### Returns

0 indicates success. Other values indicate failure.

### 7.14.5 aclfvDel

**Description**

Deletes a specified feature from a repository. Only one feature can be deleted from the repository at a time. This API is not involved in the N:M scenario.
Prototype

```c
aclError aclfvDel(aclfvFeatureInfo *featureInfo, aclrtStream stream)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>featureInfo</td>
<td>Input</td>
<td>Feature description</td>
</tr>
<tr>
<td>stream</td>
<td>Input</td>
<td>Stream</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.14.6 aclfvModify

Description

Modifies a specified feature in a repository. Only one feature in the repository can be modified at a time. This API is not involved in the N:M scenario.

Prototype

```c
aclError aclfvModify(aclfvFeatureInfo *featureInfo, aclrtStream stream)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>featureInfo</td>
<td>Input</td>
<td>Feature description</td>
</tr>
<tr>
<td>stream</td>
<td>Input</td>
<td>Stream</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.14.7 aclfvSearch

Description

Performs 1:N or N:M feature search.

Restrictions

In N:M adding mode, multithreaded adding, search, or deletion is not allowed.
Prototype

```c
aclError aclfvSearch(
    aclfvSearchType type, aclfvSearchInput *searchInput, aclfvSearchResult *searchRst, aclrtStream stream)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Input</td>
<td>Feature description</td>
</tr>
<tr>
<td>searchInput</td>
<td>Input</td>
<td>Search input</td>
</tr>
<tr>
<td>searchRst</td>
<td>Output</td>
<td>Search result</td>
</tr>
<tr>
<td>stream</td>
<td>Input</td>
<td>Stream</td>
</tr>
</tbody>
</table>

**Returns**

0 indicates success. Other values indicate failure.

## 7.15 Data Types and Operation APIs

### 7.15.1 aclError

```c
typedef int aclError;
```

**Table 7-4 Return code list**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Description</th>
<th>Possible Causes and Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>const int ACL_ERROR_NONE = 0;</td>
<td>The operation is successful.</td>
<td>-</td>
</tr>
<tr>
<td>const int ACL_ERROR_INVALID_PARAM = 100000;</td>
<td>Parameter verification failed.</td>
<td>Check the passed arguments.</td>
</tr>
<tr>
<td>const int ACL_ERROR_UNINITIALIZED = 100001;</td>
<td>ACL is not initialized.</td>
<td>Ensure that the aclInit has been called for initialization before calling other APIs.</td>
</tr>
<tr>
<td>Return Code</td>
<td>Description</td>
<td>Possible Causes and Solutions</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>const int ACL_ERROR_REPEATInicialize = 100002;</td>
<td>ACL is initialized repeatedly.</td>
<td>Check whether the aclInit is repeatedly called for initialization.</td>
</tr>
<tr>
<td>const int ACL_ERROR_INVALID_FILE = 100003;</td>
<td>The file is invalid.</td>
<td>Check whether the file exists and is accessible.</td>
</tr>
<tr>
<td>const int ACL_ERROR_WRITE_FILE = 100004;</td>
<td>File writing failed.</td>
<td>Check whether the file path exists and the write permission of the file.</td>
</tr>
<tr>
<td>const int ACL_ERROR_INVALID_FILE_SIZE = 100005;</td>
<td>The file size is invalid.</td>
<td>Check whether the file size meets the interface requirements.</td>
</tr>
<tr>
<td>const int ACL_ERROR_PARSE_FILE = 100006;</td>
<td>File parsing failed.</td>
<td>Check whether the file content is valid.</td>
</tr>
<tr>
<td>const int ACL_ERROR_FILE_MISSING_ATTR = 100007;</td>
<td>File parameter is missing.</td>
<td>Check whether the file content is complete.</td>
</tr>
<tr>
<td>const int ACL_ERROR_FILE_ATTR_INVALID = 100008;</td>
<td>File parameter is invalid.</td>
<td>Check whether the parameters in the file are correct.</td>
</tr>
<tr>
<td>const int ACL_ERROR_INVALID_DUMP_CONFIG = 100009;</td>
<td>Dump configuration is invalid.</td>
<td>Check the dump configuration in the aclInit interface configuration file. For details, see Model Accuracy Analyzer Instructions (CLI).</td>
</tr>
<tr>
<td>const int ACL_ERROR_INVALID_PROFILING_CONFIG = 100010;</td>
<td>The profiling configuration is invalid.</td>
<td>Reserved</td>
</tr>
<tr>
<td>const int ACL_ERROR_INVALID_MODEL_ID = 100011;</td>
<td>The model ID is invalid.</td>
<td>Check whether the model ID is correct and whether the model is correctly loaded.</td>
</tr>
<tr>
<td>const int ACL_ERROR_DESERIALIZATION_FAILED = 100012;</td>
<td>Model deserialization failed.</td>
<td>The model may not match the current version. Convert the model again by referring to ATC Tool Instructions.</td>
</tr>
<tr>
<td>Return Code</td>
<td>Description</td>
<td>Possible Causes and Solutions</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>const int ACL_ERROR_PARSEMODEL = 100013;</td>
<td>Model parsing failed.</td>
<td>The model may not match the current version. Convert the model again by referring to ATC Tool Instructions.</td>
</tr>
<tr>
<td>const int ACL_ERROR_READMODEL_FAILURE = 100014;</td>
<td>Model reading failed.</td>
<td>Check whether the model file exists and is accessible.</td>
</tr>
<tr>
<td>const int ACL_ERROR_MODEL_SIZEINVALID = 100015;</td>
<td>The model size is invalid.</td>
<td>The model file is invalid. Convert the model again by referring to ATC Tool Instructions.</td>
</tr>
<tr>
<td>const int ACL_ERROR_MODEL_MISSING_ATTR = 100016;</td>
<td>Model parameter is missing.</td>
<td>The model may not match the current version. Convert the model again by referring to ATC Tool Instructions.</td>
</tr>
<tr>
<td>const int ACL_ERROR_MODEL_INPUT_NOT_MATCH = 100017;</td>
<td>Model input does not match.</td>
<td>Check whether the model input is correct.</td>
</tr>
<tr>
<td>const int ACL_ERROR_MODEL_OUTPUT_NOT_MATCH = 100018;</td>
<td>Model output does not match.</td>
<td>Check whether the model output is correct.</td>
</tr>
<tr>
<td>const int ACL_ERROR_MODEL_NOT_DYNAMIC = 100019;</td>
<td>The model is non-dynamic.</td>
<td>Check whether the current model supports dynamic scenarios. If not, convert the model again by referring to ATC Tool Instructions.</td>
</tr>
<tr>
<td>const int ACL_ERROR_OP_TYPE_NOT_MATCH = 100020;</td>
<td>Single operator type does not match.</td>
<td>Check whether the operator type is correct.</td>
</tr>
<tr>
<td>const int ACL_ERROR_OP_INPUT_NOT_MATCH = 100021;</td>
<td>Single operator input does not match.</td>
<td>Check whether the operator input is correct.</td>
</tr>
<tr>
<td>const int ACL_ERROR_OP_OUTPUT_NOT_MATCH = 100022;</td>
<td>Single operator output does not match.</td>
<td>Check whether the operator output is correct.</td>
</tr>
<tr>
<td>Return Code</td>
<td>Description</td>
<td>Possible Causes and Solutions</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>const int ACL_ERROR_OP_ATTR_NOT_MATCH = 100023;</td>
<td>Single operator attribute does not match.</td>
<td>Check whether the operator attribute is correct.</td>
</tr>
<tr>
<td>const int ACL_ERROR_OP_NOT_FOUND = 100024;</td>
<td>The single operator is not found.</td>
<td>Check whether the operator type is supported.</td>
</tr>
<tr>
<td>const int ACL_ERROR_OP_LOAD_FAILED = 100025;</td>
<td>Single operator loading failed.</td>
<td>The model may not match the current version. Convert the model again by referring to ATC Tool Instructions.</td>
</tr>
<tr>
<td>const int ACL_ERROR_UNSUPPORTED_DATA_TYPE = 100026;</td>
<td>The data type is not supported.</td>
<td>Check whether the data type exists or is supported.</td>
</tr>
<tr>
<td>const int ACL_ERROR_FORMAT_NOT_MATCH = 100027;</td>
<td>The format does not match.</td>
<td>Check whether the format is correct.</td>
</tr>
<tr>
<td>const int ACL_ERROR_BIN_SELECTOR_NOT_REGISTERED = 100028;</td>
<td>The operator selector is not registered when operator is built in binary mode.</td>
<td>Check whether aclopRegisterSelectKernelFunc is called to register an operator selector.</td>
</tr>
<tr>
<td>const int ACL_ERROR_KERNEL_NOT_FOUND = 100029;</td>
<td>The operator kernel is not registered during operator building.</td>
<td>Check whether aclopCreateKernel is called to register an operator kernel.</td>
</tr>
<tr>
<td>const int ACL_ERROR_BIN_SELECTOR_ALREADY_REGISTERED = 100030;</td>
<td>The operator is repeatedly registered when the operator interface is built in binary mode.</td>
<td>Check whether aclopRegisterSelectKernelFunc is repeatedly called to register an operator selector.</td>
</tr>
<tr>
<td>const int ACL_ERROR_KERNEL_ALREADY_REGISTERED = 100031;</td>
<td>The operator kernel is repeatedly registered during operator building.</td>
<td>Check whether aclopCreateKernel is repeatedly called to register an operator kernel.</td>
</tr>
<tr>
<td>const int ACL_ERROR_INVALID_QUEUE_ID = 100032;</td>
<td>The queue ID is invalid.</td>
<td>Check whether the queue ID is correct.</td>
</tr>
<tr>
<td>Return Code</td>
<td>Description</td>
<td>Possible Causes and Solutions</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>const int ACL_ERROR_REPEAT_SUBSCRIBE = 100033;</td>
<td>The subscription is repeated.</td>
<td>Check whether the aclrtSubscribeReport is repeatedly called in the same stream.</td>
</tr>
<tr>
<td>const int ACL_ERROR_STREAM_NOT_SUBSCRIBE = 100034;</td>
<td>The stream is not subscribed to.</td>
<td>Check whether the aclrtSubscribeReport has been called.</td>
</tr>
<tr>
<td>const int ACL_ERROR_THREAD_NOT_SUBSCRIBE = 100035;</td>
<td>The thread is not subscribed to.</td>
<td>Check whether the aclrtSubscribeReport has been called.</td>
</tr>
<tr>
<td>const int ACL_ERROR_WAIT_CALLBACK_TIMEOUT = 100036;</td>
<td>The waiting for callback times out.</td>
<td>Check whether the aclrtLaunchCallback has been called to deliver callback task. Check whether the timeout period in the aclrtProcessReport is proper. Check whether the callback task has been processed. If yes, but the aclrtProcessReport is still called, the code logic needs to be optimized.</td>
</tr>
<tr>
<td>const int ACL_ERROR_REPEAT_FINALIZE = 100037;</td>
<td>Deinitialization is repeated.</td>
<td>Check whether the aclFinalize is repeatedly called for deinitialization.</td>
</tr>
<tr>
<td>const int ACL_ERROR_NOT_STATIC_AIPP = 100038;</td>
<td>The AIPP configuration information does not exist.</td>
<td>Pass the correct index to the aclmdlGetFirstAippInfo call.</td>
</tr>
<tr>
<td>const int ACL_ERROR_BAD_ALLOC = 200000;</td>
<td>Memory allocation failed.</td>
<td>Check the available memory in the hardware environment.</td>
</tr>
<tr>
<td>const int ACL_ERROR_API_NOT_SUPPORT = 200001;</td>
<td>The interface is not supported.</td>
<td>Check whether the called interface is supported.</td>
</tr>
<tr>
<td>const int ACL_ERROR_INVALID_DEVICE = 200002;</td>
<td>The device is invalid.</td>
<td>Check whether the device exists.</td>
</tr>
<tr>
<td>Return Code</td>
<td>Description</td>
<td>Possible Causes and Solutions</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>const int ACL_ERROR_MEMORY_ADDRESS_UNALIGNED = 200003;</td>
<td>The memory address is not aligned.</td>
<td>Check whether the memory address meets the interface requirements.</td>
</tr>
<tr>
<td>const int ACL_ERROR_RESOURCE_NOT_MATCH = 200004;</td>
<td>The resources do not match.</td>
<td>Check whether the input resources such as the stream and context are correctly input when the interface is called.</td>
</tr>
<tr>
<td>const int ACL_ERROR_INVALID_RESOURCE_HANDLE = 200005;</td>
<td>The resource handle is invalid.</td>
<td>Check whether the input resources such as streams and contexts have been destroyed or occupied when the interface is called.</td>
</tr>
<tr>
<td>const int ACL_ERROR_FEATURE_UNSUPPORTED = 200006;</td>
<td>The feature is not supported.</td>
<td>Rectify the fault based on the error logs, or contact Huawei technical support. For details about logs, see Log Reference.</td>
</tr>
<tr>
<td>const int ACL_ERROR_STORAGE_OVER_LIMIT = 300000;</td>
<td>The storage exceeds the upper limit.</td>
<td>Check the available disk space in the hardware environment.</td>
</tr>
<tr>
<td>const int ACL_ERROR_INTERNAL_ERROR = 500000;</td>
<td>Unknown internal error</td>
<td>Rectify the fault based on the error logs, or contact Huawei technical support. For details about logs, see Log Reference.</td>
</tr>
<tr>
<td>const int ACL_ERROR_FAILURE = 500001;</td>
<td>Internal ACL error</td>
<td>Rectify the fault based on the error logs, or contact Huawei technical support. For details about logs, see Log Reference.</td>
</tr>
<tr>
<td>const int ACL_ERROR_GE_FAILURE = 500002;</td>
<td>Internal GE error</td>
<td>Rectify the fault based on the error logs, or contact Huawei technical support. For details about logs, see Log Reference.</td>
</tr>
<tr>
<td>Return Code</td>
<td>Description</td>
<td>Possible Causes and Solutions</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>------------------------------</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>const int ACL_ERROR_RT_FAILURE = 500003;</td>
<td>Internal Runtime error</td>
<td>Rectify the fault based on the error logs, or contact Huawei technical support. For details about logs, see Log Reference.</td>
</tr>
<tr>
<td>const int ACL_ERROR_DRV_FAILURE = 500004;</td>
<td>Internal Driver error</td>
<td>Rectify the fault based on the error logs, or contact Huawei technical support. For details about logs, see Log Reference.</td>
</tr>
<tr>
<td>const int ACL_ERROR_PROFILING_FAILURE = 500005;</td>
<td>Profiling related error</td>
<td>Rectify the fault based on the error logs, or contact Huawei technical support. For details about logs, see Log Reference.</td>
</tr>
</tbody>
</table>

**NOTE**

The rules for defining return codes are as follows:

- Rule 1: Abnormal development environment or incorrect code logic that can be optimized are defined as 1XXXXX.
- Rule 2: Resource (stream, memory, etc.) insufficiency and interfaces or parameters that do not match the current hardware, which can be resolved by the proper use of resources during programming are defined as 2XXXXX.
- Rule 3: Abnormal services (full or empty queue, etc.) are defined as 3XXXXX.
- Rule 4: Abnormal internal hardware and software errors (internal software errors, device execution failure, etc.) that cannot be resolved by the user and need to be reported to Huawei are defined as 5XXXXX.
- Rule 5: Unrecognized errors are currently mapped to 500000.

### 7.15.2 aclDataType

```c
typedef enum {
    ACL_DT_UNDEFINED = -1, // Unknown data type (default)
    ACL_FLOAT = 0,
    ACL_FLOAT16 = 1,
    ACL_INT8 = 2,
    ACL_INT16 = 3,
    ACL_INT32 = 4,
    ACL_UINT8 = 5,
    ACL_UINT16 = 6,
    ACL_UINT32 = 7,
    ACL_UINT32 = 8,
    ACL_UINT64 = 9,
    ACL_UINT64 = 10,
    ACL_DOUBLE = 11,
    ACL_BOOL = 12,
} aclDataType;
```
7.15.2.1 aclDataTypeSize

Description
Obtains the size of the aclDataType data, in bytes. It is a synchronous interface.

Prototype
size_t aclDataTypeSize(aclDataType dataType)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataType</td>
<td>Input</td>
<td>Size of the aclDataType data to be obtained</td>
</tr>
</tbody>
</table>

Returns
Size of the aclDataType data, in bytes

7.15.3 aclFormat

typedef enum {
    ACL_FORMAT_UNDEFINED = -1,
    ACL_FORMAT_NCHW = 0,
    ACL_FORMAT_NHWC = 1,
    ACL_FORMAT_ND = 2,
    ACL_FORMAT_NC1HWC0 = 3,
    ACL_FORMAT_FRACTAL_Z = 4,
    ACL_FORMAT_FRACTAL_NZ = 29,
} aclFormat;

- **UNDEFINED**: Unknown format (default)
- **ND**: indicates that all formats are supported. Exercise cautions when using operators except for single-input operators that are processed inherently such as square and tanh.
- **NCHW**: NCHW format
- **NHWC**: NHWC format
- **NC1HWC0**: the 5D format defined by Huawei C0 is closely related to the micro-architecture, and the value is equal to a size of a cube unit, for example, 16. C1 divides the C dimension by C0, that is, C1 = C/C0. When the division is not exact, the last data segment is padded to C0.
- **FRACTAL_Z**: format of the convolution weight
- **FRACTAL_NZ**: format for internal use only

7.15.4 aclDvppPixelFormat

// Supported Pixel Format
enum aclDvppPixelFormat {
    PIXEL_FORMAT_YUV_400 = 0, // 0, YUV400 8bit
    PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 1, // 1, YUV420SP NV12 8bit
    PIXEL_FORMAT_YUV_SEMIPLANAR_420 = 2, // 2, YUV420SP NV21 8bit
7.15.5 aclDvppStreamFormat

```c
enum aclDvppStreamFormat {
    H265_MAIN_LEVEL = 0,
};
```
7.15.6 aclrtContext

typedef void *aclrtContext;

7.15.7 aclrtStream

typedef void *aclrtStream;

7.15.8 aclrtEvent

typedef void *aclrtEvent;

7.15.9 aclrtEventStatus

typedef enum aclrtEventStatus {
    ACL_EVENT_STATUS_COMPLETE = 0, // Completed
    ACL_EVENT_STATUS_NOT_READY = 1, // Not completed
    ACL_EVENT_STATUS_RESERVED = 2, // Reserved
} aclrtEventStatus;

7.15.10 aclTransType

typedef enum aclTransType {
    ACL_TRANS_N, // Do not transpose (default)
    ACL_TRANS_T, // Transpose
    ACL_TRANS_NZ, // Internal format, which provides better interface performance. You are advised to use
                  // this format when an input matrix is used as the repository and the operator transposed to the NZ format
                  // will be reused.
    ACL_TRANS_NZ_T // Reserved
} aclTransType;

7.15.11 aclComputeType

typedef enum aclComputeType {
    ACL_COMPUTE_HIGH_PRECISION,
    ACL_COMPUTE_LOW_PRECISION // Not supported currently
} aclComputeType;

7.15.12 aclfvsSearchType

The Ascend 310 AI Processor does not support this enum value.

typedef enum aclfvsSearchType {
    SEARCH_1_N, // 1:N operation type
    SEARCH_N_M  // N:M operation type
};

7.15.13 aclAippInputFormat

typedef enum {
    ACL_YUV420SP_U8 = 1, //YUV420SP_U8
    ACL_XRGB8888_U8=2, //XRGB8888_U8
    ACL_RGB888_U8=3, //RGB888_U8
    ACL_YUV400_U8=4, //YUV400_U8
    /*Dynamic AIPP and static AIPP support the preceding four enum values.*/
} aclAippInputFormat;
typedef struct aclAippInfo {
    aclAippInputFormat inputFormat;
    int32_t srcImageSizeW;
    int32_t srcImageSizeH;
    int8_t cropSwitch;
    int32_t loadStartPosW;
    int32_t loadStartPosH;
    int32_t cropSizeW;
    int32_t cropSizeH;
    int8_t resizeSwitch;
    int32_t resizeOutputW;
    int32_t resizeOutputH;
    int8_t paddingSwitch;
    int32_t leftPaddingSize;
    int32_t rightPaddingSize;
    int32_t topPaddingSize;
    int32_t bottomPaddingSize;
    int8_t cscSwitch;
    int8_t rbuvSwapSwitch;
    int8_t axSwapSwitch;
    int8_t singleLineMode;
    int32_t matrixR0C0;
    int32_t matrixR0C1;
    int32_t matrixR0C2;
    int32_t matrixR1C0;
    int32_t matrixR1C1;
    int32_t matrixR1C2;
    int32_t matrixR2C0;
    int32_t matrixR2C1;
    int32_t matrixR2C2;
    int32_t outputBias0;
    int32_t outputBias1;
    int32_t outputBias2;
    int32_t inputBias0;
    int32_t inputBias1;
    int32_t inputBias2;
    int32_t meanChn0;
    int32_t meanChn1;
    int32_t meanChn2;
    int32_t meanChn3;
    float minChn0;
    float minChn1;
    float minChn2;
    float minChn3;
    float varReciChn0;
    float varReciChn1;
    float varReciChn2;
    float varReciChn3;
    aclFormat srcFormat; // Source format
    aclDataType srcDatatype; // Source data type
    size_t srcDimNum; // Source dimension count
    size_t shapeCount; // Number of batch/image size choices in the dynamic shape scenario
    aclAippDims outDims[ACL_MAX_SHAPE_COUNT];
} aclAippInputFormat;
### 7.15.15 aclAippDims

typedef struct aclAippDims {
    aclmdlIODims srcDims; // Source dimensions, for example, [1, 3, 150, 150]
    size_t srcSize; // Source data size
    aclmdlIODims aippOutdims; // Destination dimensions
    size_t aippOutSize; // Destination data size
} aclAippDims;

### 7.15.16 aclmdlIODims

const int ACL_MAX_DIM_CNT = 128;
const int ACL_MAX_TENSOR_NAME_LEN = 128;
typedef struct aclmdlIODims {
    char name[ACL_MAX_TENSOR_NAME_LEN]; /**< tensor name */
    size_t dimCount; /**< Dimension count*/
    int64_t dims[ACL_MAX_DIM_CNT]; /**< Dimension information*/
} aclmdlIODims;

### 7.15.17 aclmdlAIPP

Interfaces of this data type are all synchronous interfaces.

#### 7.15.17.1 aclmdlCreateAIPP

**Description**

Creates data of the `aclmdlCreateAIPP` type based on the batch size supported by the model in dynamic AIPP scenarios, to store dynamic AIPP parameters.

**Prototype**

```c
aclmdlAIPP *aclmdlCreateAIPP(uint64_t batchSize)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>batchSize</td>
<td>Input</td>
<td>Batch size of the model</td>
</tr>
</tbody>
</table>

**Returns**

`aclmdlAIPP` address

#### 7.15.17.2 Dynamic AIPP Parameters Setting
7.15.17.2.1 aclmdlSetAIPPInputFormat

**Description**
Sets the input format of the source image in `dynamic AIPP` scenarios. It is a synchronous interface.

**Prototype**
```c
aclError aclmdlSetAIPPInputFormat(aclmdlAIPP *aippParmsSet, aclAippInputFormat inputFormat)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aippParmsSet</td>
<td>Input</td>
<td>Dynamic AIPP parameter object. Call <code>aclmdlCreateAIPP</code> in advance to create data of the <code>aclmdlAIPP</code> type.</td>
</tr>
<tr>
<td>inputFormat</td>
<td>Input</td>
<td>Format of the source image. Format range is as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>ACL_YUV420 SP_U8</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>ACL_XRGB8888_U8</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>ACL_RGB888_U8</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <code>ACL_YUV400_U8</code></td>
</tr>
</tbody>
</table>

**Returns**
0 indicates success. Other values indicate failure.

7.15.17.2.2 aclmdlSetAIPPCscParams

**Description**
Sets CSC parameters in `dynamic AIPP` scenarios. If the CSC switch is disabled, parameters set by calling this interface are invalid. It is a synchronous interface.

YUV2BGR conversion:
```
| B | | cscMatrixR0C0 cscMatrixR0C1 cscMatrixR0C2 | | Y - cscInputBiasR0 |
| G | = | cscMatrixR1C0 cscMatrixR1C1 cscMatrixR1C2 | | U - cscInputBiasR1 | >> 8 |
| R | | cscMatrixR2C0 cscMatrixR2C1 cscMatrixR2C2 | | V - cscInputBiasR2 |
```

BGR2YUV conversion:
```
| Y | | cscMatrixR0C0 cscMatrixR0C1 cscMatrixR0C2 | | B | | cscOutputBiasR0 |
| U | = | cscMatrixR1C0 cscMatrixR1C1 cscMatrixR1C2 | | G | >> 8 + | cscOutputBiasR1 |
| V | | cscMatrixR2C0 cscMatrixR2C1 cscMatrixR2C2 | | R | | cscOutputBiasR2 |
```

CSC parameters are closely related to the formats of the source and destination images. You can refer to **CSC Configuration** in ATC Tool Instructions for CSC parameters configuration. Configure CSC parameters as required if the listed image formats do not meet your requirements.
Prototype

```c
aclError aclmdlSetAIPPScsParams(aclmdlAIPP *aippParmsSet, int8_t csc_switch,
int16_t cscMatrixR0C0, int16_t cscMatrixR0C1, int16_t cscMatrixR0C2,
int16_t cscMatrixR1C0, int16_t cscMatrixR1C1, int16_t cscMatrixR1C2,
int16_t cscMatrixR2C0, int16_t cscMatrixR2C1, int16_t cscMatrixR2C2,
uint8_t cscOutputBiasR0, uint8_t cscOutputBiasR1, uint8_t cscOutputBiasR2,
uint8_t cscInputBiasR0, uint8_t cscInputBiasR1, uint8_t cscInputBiasR2)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aippParmsSet</td>
<td>Input</td>
<td>Dynamic AIPP parameter object</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Call <code>aclmdlCreateAIPP</code> in advance to create data of the <code>aclmdlAIPP</code> type.</td>
</tr>
<tr>
<td>csc_switch</td>
<td>Input</td>
<td>CSC switch</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 0 (default): disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 1: enabled</td>
</tr>
<tr>
<td>cscMatrixR0C0</td>
<td>Input</td>
<td>CSC matrix parameter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value range: [-32677, +32676]</td>
</tr>
<tr>
<td>cscMatrixR0C1</td>
<td>Input</td>
<td>CSC matrix parameter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value range: [-32677, +32676]</td>
</tr>
<tr>
<td>cscMatrixR0C2</td>
<td>Input</td>
<td>CSC matrix parameter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value range: [-32677, +32676]</td>
</tr>
<tr>
<td>cscMatrixR1C0</td>
<td>Input</td>
<td>CSC matrix parameter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value range: [-32677, +32676]</td>
</tr>
<tr>
<td>cscMatrixR1C1</td>
<td>Input</td>
<td>CSC matrix parameter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value range: [-32677, +32676]</td>
</tr>
<tr>
<td>cscMatrixR1C2</td>
<td>Input</td>
<td>CSC matrix parameter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value range: [-32677, +32676]</td>
</tr>
<tr>
<td>cscMatrixR2C0</td>
<td>Input</td>
<td>CSC matrix parameter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value range: [-32677, +32676]</td>
</tr>
<tr>
<td>cscMatrixR2C1</td>
<td>Input</td>
<td>CSC matrix parameter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value range: [-32677, +32676]</td>
</tr>
<tr>
<td>Parameter</td>
<td>Input/Output</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>cscMatrixR2C2</td>
<td>Input</td>
<td>CSC matrix parameter Value range: [-32677, +32676]</td>
</tr>
<tr>
<td>cscOutputBiasR0</td>
<td>Input</td>
<td>Output bias during RGB2YUV conversion. The default value is 0 Value range: [0, 255]</td>
</tr>
<tr>
<td>cscOutputBiasR1</td>
<td>Input</td>
<td>Output bias during RGB2YUV conversion. The default value is 0 Value range: [0, 255]</td>
</tr>
<tr>
<td>cscOutputBiasR2</td>
<td>Input</td>
<td>Output bias during RGB2YUV conversion. The default value is 0 Value range: [0, 255]</td>
</tr>
<tr>
<td>cscInputBiasR0</td>
<td>Input</td>
<td>Input bias during YUV2RGB conversion. The default value is 0 Value range: [0, 255]</td>
</tr>
<tr>
<td>cscInputBiasR1</td>
<td>Input</td>
<td>Input bias during YUV2RGB conversion. The default value is 0 Value range: [0, 255]</td>
</tr>
<tr>
<td>cscInputBiasR2</td>
<td>Input</td>
<td>Input bias during YUV2RGB conversion. The default value is 0 Value range: [0, 255]</td>
</tr>
</tbody>
</table>

**Returns**

0 indicates success. Other values indicate failure.

### 7.15.17.2.3 aclmdlSetAIPPRbuvSwapSwitch

**Description**

Sets the RB or UV swap switch in dynamic AIPP scenarios. It is a synchronous interface.

**Prototype**

```c
aclError aclmdlSetAIPPRbuvSwapSwitch(aclmdlAIPP *aippParmsSet, int8_t rbuvSwapSwitch)
```
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
</table>
| aippParmsSet     | Input        | Dynamic AIPP parameter object  
                           Call `aclmdlCreateAIPP` in advance to create data of the `aclmdlAIPP` type. |
| rbuvSwapSwitch   | Input        | RB or UV channel swap switch  
                           ● 0 (default): disabled  
                           ● 1: enabled |

#### Returns

0 indicates success. Other values indicate failure.

### 7.15.17.2.4 aclmdlSetAIPPAxSwapSwitch

#### Description

Sets the RGBA-to-ARGB or YUVA-to-AYUV swap switch in dynamic AIPP scenarios. It is a synchronous interface.

#### Prototype

```
aclError aclmdlSetAIPPAxSwapSwitch(aclmdlAIPP *aippParmsSet, int8_t axSwapSwitch)
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
</table>
| aippParmsSet     | Input        | Dynamic AIPP parameter object  
                           Call `aclmdlCreateAIPP` in advance to create data of the `aclmdlAIPP` type. |
| axSwapSwitch     | Input        | RGBA to ARGB, or YUVA to AYUV swap switch  
                           ● 0 (default): disabled  
                           ● 1: enabled |

#### Returns

0 indicates success. Other values indicate failure.
7.15.17.2.5 aclmdlSetAIPPSrcImageSize

**Description**

Sets the size of the source image in dynamic AIPP scenarios. It is a synchronous interface.

**Prototype**

```c
aclError aclmdlSetAIPPSrcImageSize(aclmdlAIPP *aippParmsSet, int32_t srcImageSizeW, int32_t srcImageSizeH)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aippParmsSet</td>
<td>Input</td>
<td>Dynamic AIPP parameter object. Call aclmdlCreateAIPP in advance to create data of the aclmdlAIPP type.</td>
</tr>
<tr>
<td>srcImageSizeW</td>
<td>Input</td>
<td>Source image width. For the YUV420 SP_U8 format, the value must be an even number. Value range: [0, 4096]</td>
</tr>
<tr>
<td>srcImageSizeH</td>
<td>Input</td>
<td>Source image height. For the YUV420 SP_U8 format, the value must be an even number. Value range: [0, 4096]</td>
</tr>
</tbody>
</table>

**Returns**

0 indicates success. Other values indicate failure.

7.15.17.2.6 aclmdlSetAIPPScfParams

**Description**

Sets resizing parameters in dynamic AIPP scenarios. It is a synchronous interface. The current version does not support this interface.

**Prototype**

```c
aclError aclmdlSetAIPPScfParams(aclmdlAIPP *aippParmsSet, int8_t scfSwitch, int32_t scfInputSizeW, int32_t scfInputSizeH, int32_t scfOutputSizeW, int32_t scfOutputSizeH, uint64_t batchIndex)
```
Restrictions

The resizing ratio is $\frac{\text{scfOutputSizeW}}{\text{scfInputSizeW}} \in [1/16, 16]$ or $\frac{\text{scfOutputSizeH}}{\text{scfInputSizeH}} \in [1/16, 16]$.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aippParmsSet</td>
<td>Input</td>
<td>Dynamic AIPP parameter object</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Call aclmdlCreateAIPP in advance to create data of the aclmdlAIPP type.</td>
</tr>
<tr>
<td>scfSwitch</td>
<td>Input</td>
<td>Resizing enable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 0 (default): disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 1: enabled</td>
</tr>
<tr>
<td>scfInputSizeW</td>
<td>Input</td>
<td>Source image width</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value range: [0, 4096]</td>
</tr>
<tr>
<td>scfInputSizeH</td>
<td>Input</td>
<td>Source image height</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value range: [0, 4096]</td>
</tr>
<tr>
<td>scfOutputSizeW</td>
<td>Input</td>
<td>Destination image width</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value range: [0, 4096]</td>
</tr>
<tr>
<td>scfOutputSizeH</td>
<td>Input</td>
<td>Destination image height</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value range: [0, 4096]</td>
</tr>
<tr>
<td>batchIndex</td>
<td>Input</td>
<td>Index of batch for image resizing. Defaults to 0.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value range: [0, batchSize]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>batchSize is set when data of the aclmdlAIPP type is created by calling the aclmdlCreateAIPP API.</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.15.17.2.7 aclmdlSetAIPPCropParams

Description

Sets cropping parameters in dynamic AIPP scenarios. It is a synchronous interface.
Prototype

```c
acleError aclmdlSetAIPPCropParams(aclmdlAIPP *aippParmsSet, int8_t cropSwitch,
                                   int32_t cropStartPosW, int32_t cropStartPosH,
                                   int32_t cropSizeW, int32_t cropSizeH,
                                   uint64_t batchIndex)
```

Restrictions

If the cropping function is enabled, the following conditions must be met for the arguments passed to the `aclmdlSetAIPPSrcImageSize` call and `aclmdlSetAIPPCropParams` call:

- `srcImageSizeW ≥ cropSizeW + cropStartPosW`
- `srcImageSizeH ≥ cropSizeH + cropStartPosH`

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aippParmsSet</td>
<td>Input</td>
<td>Dynamic AIPP parameter object Call <code>aclmdlCreateAIPP</code> in advance to create data of the <code>aclmdlAIPP</code> type.</td>
</tr>
<tr>
<td>cropSwitch</td>
<td>Input</td>
<td>Cropping enable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 0 (default): disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 1: enabled</td>
</tr>
<tr>
<td>cropStartPosW</td>
<td>Input</td>
<td>Horizontal coordinate of the crop start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For YUV420 SP_U8, the value must be an even number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value range: [0, 4096]</td>
</tr>
<tr>
<td>cropStartPosH</td>
<td>Input</td>
<td>Vertical coordinate of the crop start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For YUV420 SP_U8, the value must be an even number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value range: [0, 4096]</td>
</tr>
<tr>
<td>cropSizeW</td>
<td>Input</td>
<td>Width of the cropping area</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For YUV420 SP_U8, the value must be an even number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value range: [0, 4096]</td>
</tr>
</tbody>
</table>
### Parameter Table

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cropSizeH</td>
<td>Input</td>
<td>Height of the cropping area For YUV420 SP_U8, the value must be an even number. Value range: [0, 4096]</td>
</tr>
</tbody>
</table>
| batchIndex   | Input        | Index of batch for image cropping. The default value is 0. Value range: [0, batchSize]  
batchSize is set when data of the aclmdlAIPP type is created by calling the aclmdlCreateAIPP API. |

### Returns

0 indicates success. Other values indicate failure.

### 7.15.17.2.8 aclmdlSetAIPPPaddingParams

**Description**

Sets padding parameters in dynamic AIPP scenarios. It is a synchronous interface.

**Prototype**

```
acError aclmdlSetAIPPPaddingParams(aclmdlAIPP *aippParmsSet, int8_t paddingSwitch,
int32_t paddingSizeTop, int32_t paddingSizeBottom,
int32_t paddingSizeLeft, int32_t paddingSizeRight,
uint64_t batchIndex)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aippParmsSet</td>
<td>Input</td>
<td>Dynamic AIPP parameter object Call aclmdlCreateAIPP in advance to create data of the aclmdlAIPP type.</td>
</tr>
</tbody>
</table>
| paddingSwitch| Input        | Padding switch  
• 0 (default): disabled  
• 1: enabled                                                             |
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>paddingSizeTop</td>
<td>Input</td>
<td>Top padding size</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value range: [0, 32]</td>
</tr>
<tr>
<td>paddingSizeBottom</td>
<td>Input</td>
<td>Bottom padding size</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value range: [0, 32]</td>
</tr>
<tr>
<td>paddingSizeLeft</td>
<td>Input</td>
<td>Left padding size</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value range: [0, 32]</td>
</tr>
<tr>
<td>paddingSizeRight</td>
<td>Input</td>
<td>Right padding size</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value range: [0, 32]</td>
</tr>
<tr>
<td>batchIndex</td>
<td>Input</td>
<td>Index of batch for image padding. The default value is 0.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value range: [0, batchSize]</td>
</tr>
</tbody>
</table>

**Returns**

0 indicates success. Other values indicate failure.

### 7.15.17.2.9 aclmdlSetAIPPDtcPixelMean

#### Description

Sets the mean value of each channel in dynamic AIPP scenarios. It is a synchronous interface.

#### Prototype

```c
aclError aclmdlSetAIPPDtcPixelMean(aclmdlAIPP *aippParmsSet,
int16_t dtcPixelMeanChn0,
int16_t dtcPixelMeanChn1,
int16_t dtcPixelMeanChn2,
int16_t dtcPixelMeanChn3,
uint64_t batchIndex)
```
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aippParmsSet</td>
<td>Input</td>
<td>Dynamic AIPP parameter object Call <a href="#">aclmdlCreateAIPP</a> in advance to create data of the aclmdlAIPP type.</td>
</tr>
<tr>
<td>dtcPixelMeanChn0</td>
<td>Input</td>
<td>Mean value of channel 0 Value range: [0, 255]</td>
</tr>
<tr>
<td>dtcPixelMeanChn1</td>
<td>Input</td>
<td>Mean value of channel 1 Value range: [0, 255]</td>
</tr>
<tr>
<td>dtcPixelMeanChn2</td>
<td>Input</td>
<td>Mean value of channel 2 Value range: [0, 255]</td>
</tr>
<tr>
<td>dtcPixelMeanChn3</td>
<td>Input</td>
<td>Mean value of channel 3 If there are only three channels, the default value is 0. Value range: [0, 255]</td>
</tr>
<tr>
<td>batchIndex</td>
<td>Input</td>
<td>Index of batch for channel mean value setting. The default value is 0. Value range: [0, batchSize] batchSize is set when data of the aclmdlAIPP type is created by calling the aclmdlCreateAIPP API.</td>
</tr>
</tbody>
</table>

### Returns

0 indicates success. Other values indicate failure.

#### 7.15.17.2.10 aclmdlSetAIPPDtcPixelMin

### Description

Sets the minimum value of each channel in dynamic AIPP scenarios. It is a synchronous interface.

### Prototype

```c
aclError aclmdlSetAIPPDtcPixelMin(aclmdlAIPP *aippParmsSet,
                                 float dtcPixelMinChn0,
                                 float dtcPixelMinChn1,
                                 float dtcPixelMinChn2,
                                 float dtcPixelMinChn3,
```


uint64_t batchIndex)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aippParmsSet</td>
<td>Input</td>
<td>Dynamic AIPP parameter object. Call aclmdlCreateAIPP in advance to create data of the aclmdlAIPP type.</td>
</tr>
<tr>
<td>dtcPixelMinChn0</td>
<td>Input</td>
<td>Minimum value of channel 0. Value range: [0, 255]</td>
</tr>
<tr>
<td>dtcPixelMinChn1</td>
<td>Input</td>
<td>Minimum value of channel 1. Value range: [0, 255]</td>
</tr>
<tr>
<td>dtcPixelMinChn2</td>
<td>Input</td>
<td>Minimum value of channel 2. Value range: [0, 255]</td>
</tr>
<tr>
<td>dtcPixelMinChn3</td>
<td>Input</td>
<td>Minimum value of channel 3. If there are only three channels, the default value is 0. Value range: [0, 255]</td>
</tr>
<tr>
<td>batchIndex</td>
<td>Input</td>
<td>Index of batch for channel minimum value setting. The default value is 0. Value range: [0, batchSize]</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.15.17.2.11 aclmdlSetAIPPPixelVarReci

Description

Sets the variance of each channel in dynamic AIPP scenarios. It is a synchronous interface.

Prototype

aclError aclmdlSetAIPPPixelVarReci(aclmdlAIPP *aippParmsSet,
float dtcPixelVarReciChn0,
float dtcPixelVarReciChn1,


```c
float dtcPixelVarReciChn2,
float dtcPixelVarReciChn3,
uint64_t batchIndex)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aippParmsSet</td>
<td>Input</td>
<td>Dynamic AIPP parameter object. Call <code>aclmdlCreateAIPP</code> in advance to create data of the <code>aclmdlAIPP</code> type.</td>
</tr>
<tr>
<td>dtcPixelVarReciChn0</td>
<td>Input</td>
<td>Variance of channel 0. Value range: [-65504, +65504]</td>
</tr>
<tr>
<td>dtcPixelVarReciChn1</td>
<td>Input</td>
<td>Variance of channel 1. Value range: [-65504, +65504]</td>
</tr>
<tr>
<td>dtcPixelVarReciChn2</td>
<td>Input</td>
<td>Variance of channel 2. Value range: [-65504, +65504]</td>
</tr>
<tr>
<td>dtcPixelVarReciChn3</td>
<td>Input</td>
<td>Variance of channel 3. If there are only three channels, the default value is 1.0. Value range: [-65504, +65504]</td>
</tr>
<tr>
<td>batchIndex</td>
<td>Input</td>
<td>Index of batch for channel variance setting. The default value is 0. Value range: [0, batchSize]</td>
</tr>
</tbody>
</table>

### Returns

0 indicates success. Other values indicate failure.

### 7.15.17.3 aclmdlDestroyAIPP

#### Description

Destroys data of the `aclmdlAIPP` type.

#### Prototype

```c
aclError aclmdlDestroyAIPP(const aclmdlAIPP *aippParmsSet)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aippParmsSet</td>
<td>Input</td>
<td>Pointer to the aclmdlAIPP to be destroyed</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.15.18 aclopHandle

Interfaces of this data type are all synchronous interfaces.

7.15.18.1 aclopCreateHandle

Description

Creates a handle to an execution operator.

Prototype

```c
aclError aclopCreateHandle(const char *opType,
int numInputs,
const aclTensorDesc *const inputDesc[],
int numOutputs,
const aclTensorDesc *const outputDesc[],
const aclopAttr *opAttr,
aclopHandle **handle);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>opType</td>
<td>Input</td>
<td>Operator type</td>
</tr>
<tr>
<td>numInputs</td>
<td>Input</td>
<td>Number of input tensors</td>
</tr>
<tr>
<td>inputDesc</td>
<td>Input</td>
<td>Input tensor description</td>
</tr>
<tr>
<td>numOutputs</td>
<td>Input</td>
<td>Number of output tensors</td>
</tr>
<tr>
<td>outputDesc</td>
<td>Input</td>
<td>Output tensor description</td>
</tr>
<tr>
<td>opAttr</td>
<td>Input</td>
<td>Operator attributes</td>
</tr>
<tr>
<td>Parameter</td>
<td>Input/Output</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>handle</td>
<td>Output</td>
<td>Pointer to the pointer to the aclopHandle data</td>
</tr>
</tbody>
</table>

**Returns**

0 indicates success. Other values indicate failure.

### 7.15.18.2 aclopDestroyHandle

**Description**

Destroys the handle to an execution operator.

**Prototype**

```c
void aclopDestroyHandle(aclopHandle *handle)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>Input</td>
<td>Pointer to the aclopHandle to be destroyed</td>
</tr>
</tbody>
</table>

**Returns**

None

### 7.15.19 aclFloat16

```c
typedef uint16_t aclFloat16;
```

Interfaces of this data type are all synchronous interfaces.

### 7.15.19.1 aclFloat16ToFloat

**Description**

Converts data from aclFloat16 to float32.

**Prototype**

```c
float aclFloat16ToFloat(aclFloat16 value)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Input</td>
<td>Data to be converted</td>
</tr>
</tbody>
</table>

Returns

Data after conversion

7.15.19.2 aclFloatToFloat16

Description

Converts data from float32 to aclFloat16.

Prototype

```
calFloat16 aclFloatToFloat16(float value)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Input</td>
<td>Data to be converted</td>
</tr>
</tbody>
</table>

Returns

Data after conversion

7.15.20 aclDataBuffer

The following interfaces can be called to obtain data of the aclDataBuffer type and perform relevant operations on the data. Interfaces of this data type are all synchronous interfaces.

aclDataBuffer is used to describe memory information such as the memory address and size.

7.15.20.1 aclCreateDataBuffer

Description

Creates data of the aclDataBuffer type.

Prototype

```
claDataBuffer *aclCreateDataBuffer(void *data, size_t size)
```
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Input</td>
<td>Memory address&lt;br&gt;The memory needs to be managed by the user. Call <code>aclrtMalloc</code> to allocate memory and call <code>aclrtFree</code> to free memory.</td>
</tr>
<tr>
<td>size</td>
<td>Input</td>
<td>Memory size, in bytes&lt;br&gt;If an empty tensor is needed, the minimum value to be set is 1 byte.</td>
</tr>
</tbody>
</table>

### Returns

`aclDataBuffer address`

#### 7.15.20.2 aclDestroyDataBuffer

**Description**

Destroys data of the `aclDataBuffer` type.

Destroys only data of the `aclDataBuffer` type. When `aclCreateDataBuffer` is called to create data of the `aclDataBuffer` type, the input data memory needs to be freed by the user.

**Prototype**

```
aplError aclDestroyDataBuffer(const aclDataBuffer *dataBuffer)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataBuffer</td>
<td>Input</td>
<td>Pointer to the <code>aclDataBuffer</code> to be destroyed</td>
</tr>
</tbody>
</table>

**Returns**

0 indicates success. Other values indicate failure.

#### 7.15.20.3 aclGetDataBufferAddr

**Description**

Obtains memory address of data of the `aclDataBuffer` type.
Prototype

```c
void *aclGetDataBufferAddr(const aclDataBuffer *dataBuffer)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataBuffer</td>
<td>Input</td>
<td>aclDataBuffer address</td>
</tr>
</tbody>
</table>

Returns

None

7.15.20.4 aclGetDataBufferSize

Description

Obtains memory size of data of the aclDataBuffer type, in bytes.

Prototype

```c
size_t aclGetDataBufferSize(const aclDataBuffer *dataBuffer)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataBuffer</td>
<td>Input</td>
<td>aclDataBuffer address</td>
</tr>
</tbody>
</table>

Returns

Memory size of data of the aclDataBuffer type

7.15.21 aclmdlDataset

Calls the following interfaces to obtain data of the aclmdlDataset type and perform relevant operations on the data. Interfaces of this data type are all synchronous interfaces.

aclmdlDataset is used to describe input and output data during model inference. A model may have multiple inputs and outputs, and data of the aclDataBuffer type is used to describe the memory address and memory size of each input or output.
7.15.21.1 aclmdlCreateDataset

**Description**

Creates data of the `aclmdlDataset` type.

**Prototype**

```c
aclmdlDataset *aclmdlCreateDataset()
```

**Parameters**

None

**Returns**

`aclmdlDataset` address

7.15.21.2 aclmdlDestroyDataset

**Description**

Destroys data of the `aclmdlDataset` type.

**Prototype**

```c
aclError aclmdlDestroyDataset(const aclmdlDataset *dataset)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataset</td>
<td>Input</td>
<td>Pointer to the <code>aclmdlDataset</code> to be destroyed</td>
</tr>
</tbody>
</table>

**Returns**

0 indicates success. Other values indicate failure.

7.15.21.3 aclmdlAddDatasetBuffer

**Description**

Adds `aclDataBuffer` to `aclmdlDataset`.

**Prototype**

```c
aclError aclmdlAddDatasetBuffer(aclmdlDataset *dataset, aclDataBuffer *dataBuffer)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataset</td>
<td>Input/Output</td>
<td>aclmdlDataset address where aclDataBuffer is added</td>
</tr>
<tr>
<td>dataBuffer</td>
<td>Input</td>
<td>aclDataBuffer address</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.15.21.4 aclmdlGetDatasetNumBuffers

Description

Obtains the number of aclDataBuffers in aclmdlDataset.

Prototype

```
size_t aclmdlGetDatasetNumBuffers(const aclmdlDataset *dataset)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataset</td>
<td>Input</td>
<td>aclmdlDataset address</td>
</tr>
</tbody>
</table>

Returns

Number of aclDataBuffers

7.15.21.5 aclmdlGetDatasetBuffer

Description

Obtains the n<sup>th</sup> aclDataBuffer in aclmdlDataset.

Prototype

```
aclDataBuffer* aclmdlGetDatasetBuffer(const aclmdlDataset *dataset, size_t index)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataset</td>
<td>Input</td>
<td>aclmdlDataset address</td>
</tr>
<tr>
<td>index</td>
<td>Input</td>
<td>Sequence number of the aclDataBuffer to be obtained</td>
</tr>
</tbody>
</table>

Returns

- Success: aclDataBuffer address
- Failure: empty address

7.15.22 aclmdlDesc

Interfaces of this data type are all synchronous interfaces.

7.15.22.1 aclmdlCreateDesc

Description

Creates data of the aclmdlDesc type.

Prototype

```c
aclmdlDesc* aclmdlCreateDesc()
```

Parameters

None

Returns

aclmdlDesc address

7.15.22.2 aclmdlDestroyDesc

Description

Destroys data of the aclmdlDesc type.

Prototype

```c
aclError aclmdlDestroyDesc(aclmdlDesc *modelDesc)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelDesc</td>
<td>Input</td>
<td>Pointer to the aclmdlDesc to be destroyed</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.15.22.3 aclmdlGetDesc

Description

Obtains model data of the aclmdlDesc type using the model ID.

Prototype

aclError aclmdlGetDesc(aclmdlDesc *modelDesc, uint32_t modelId)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelDesc</td>
<td>Output</td>
<td>Pointer to the data of the aclmdlDesc type</td>
</tr>
<tr>
<td>modelId</td>
<td>Input</td>
<td>Model ID</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.15.22.4 aclmdlGetNumInputs

Description

Obtains the number of input models based on data of the aclmdlDesc type.

Prototype

size_t aclmdlGetNumInputs(aclmdlDesc *modelDesc)
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelDesc</td>
<td>Input</td>
<td>Pointer to the data of the aclmdlDesc type</td>
</tr>
</tbody>
</table>

### Returns

- Number of input models

### 7.15.22.5 aclmdlGetNumOutputs

#### Description

Obtains the number of output models based on data of the aclmdlDesc type.

#### Prototype

```c
size_t aclmdlGetNumOutputs(aclmdlDesc *modelDesc)
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelDesc</td>
<td>Input</td>
<td>Pointer to the data of the aclmdlDesc type</td>
</tr>
</tbody>
</table>

### Returns

- Number of output models

### 7.15.22.6 aclmdlGetInputSizeByIndex

#### Description

Obtains the size of an input by index based on data of the aclmdlDesc type. The unit is byte.

#### Prototype

```c
size_t aclmdlGetInputSizeByIndex(aclmdlDesc *modelDesc, size_t index)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelDesc</td>
<td>Input</td>
<td>Pointer to the data of the aclmdlDesc type</td>
</tr>
<tr>
<td>index</td>
<td>Input</td>
<td>Sequence number of the input to be obtained, indexed starting at 0</td>
</tr>
</tbody>
</table>

Returns

If dynamic batch/image size is enabled, the input size of the maximum choice is returned. In dynamic batch/image size scenarios, the specified input size is returned. The unit is byte.

7.15.22.7 aclmdlGetOutputSizeByIndex

Description

Obtains the size of an output by index based on data of the aclmdlDesc type. The unit is byte.

Prototype

```c
size_t aclmdlGetOutputSizeByIndex(aclmdlDesc *modelDesc, size_t index)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelDesc</td>
<td>Input</td>
<td>Pointer to the data of the aclmdlDesc type</td>
</tr>
<tr>
<td>index</td>
<td>Input</td>
<td>Sequence number of the output to be obtained, indexed starting at 0</td>
</tr>
</tbody>
</table>

Returns

If dynamic batch/image size is enabled, the output size of the maximum choice is returned. In dynamic batch/image size scenarios, the specified output size is returned. The unit is byte.

7.15.22.8 aclmdlGetInputDims

Description

Obtains dimension information of the input tensor based on the model description.
Prototype

```c
aclError aclmdlGetInputDims(const aclmdlDesc *modelDesc, size_t index,
aclmdlIODims *dims)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelDesc</td>
<td>Input</td>
<td>Pointer to the data of the aclmdlDesc type</td>
</tr>
<tr>
<td>index</td>
<td>Input</td>
<td>Sequence number of the input whose Dims is to be obtained, indexed starting at 0</td>
</tr>
<tr>
<td>dims</td>
<td>Output</td>
<td>Obtains the input dimension information. In dynamic batch/image size scenarios, if the batch size, or the width and height in the input tensor are -1, the shape is dynamic. For example, if the format of the input tensor is NCHW, then the tensor dimension [-1,3,224,224] indicates in the batch size is dynamic, and the tensor dimension [7,3,-1,-1] indicates the image size is dynamic. The italic numbers are examples only.</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.15.22.9 aclmdlGetOutputDims

Description

Obtains dimension information of the output tensor based on the model description.

In the fixed shape scenario, this API is used to obtain the dimension information of the output tensor of a model.

This API is used to obtain the dimension information of the maximum batch/image size choice in the dynamic shape scenario.

Prototype

```c
aclError aclmdlGetOutputDims(const aclmdlDesc *modelDesc, size_t index,
aclmdlIODims *dims)
```
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelDesc</td>
<td>Input</td>
<td>Pointer to the data of the aclmdlDesc type</td>
</tr>
<tr>
<td>index</td>
<td>Input</td>
<td>Sequence number of the output whose dims information is to be obtained, indexed starting at 0</td>
</tr>
<tr>
<td>dims</td>
<td>Output</td>
<td>Input dimension information</td>
</tr>
</tbody>
</table>

### Returns

0 indicates success. Other values indicate failure.

#### 7.15.22.10 aclmdlGetInputNameByIndex

### Description

Obtains the name of an input in a model.

### Prototype

```c
const char *aclmdlGetInputNameByIndex(const aclmdlDesc *modelDesc, size_t index)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelDesc</td>
<td>Input</td>
<td>Pointer to the data of the aclmdlDesc type</td>
</tr>
<tr>
<td>index</td>
<td>Input</td>
<td>Sequence number of the input whose name is to be obtained, indexing starting at 0</td>
</tr>
</tbody>
</table>

### Returns

The input name

#### 7.15.22.11 aclmdlGetOutputNameByIndex

### Description

Obtains the name of an output of a model and the index of the edge.
Prototype

```c
const char *aclmdlGetOutputNameByIndex(const aclmdlDesc *modelDesc, size_t index)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelDesc</td>
<td>Input</td>
<td>Pointer to the data of the <code>aclmdlDesc</code> type</td>
</tr>
<tr>
<td>index</td>
<td>Input</td>
<td>Sequence number of the output, indexed starting at 0</td>
</tr>
</tbody>
</table>

Returns

The output name and edge index

7.15.22.12 aclmdlGetInputFormat

Description

Obtains the format of an input in a model.

Prototype

```c
aclFormat aclmdlGetInputFormat(const aclmdlDesc *modelDesc, size_t index)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelDesc</td>
<td>Input</td>
<td>Pointer to the data of the <code>aclmdlDesc</code> type</td>
</tr>
<tr>
<td>index</td>
<td>Input</td>
<td>Sequence number of the input whose format is to be obtained, indexing starting at 0</td>
</tr>
</tbody>
</table>

Returns

The format of the specified input

7.15.22.13 aclmdlGetOutputFormat

Description

Obtains the format of an output in a model.
Prototype

```
aclFormat aclmdlGetOutputFormat(const aclmdlDesc *modelDesc, size_t index)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelDesc</td>
<td>Input</td>
<td>Pointer to the data of the <code>aclmdlDesc</code> type</td>
</tr>
<tr>
<td>index</td>
<td>Input</td>
<td>Sequence number of the output whose format is to be obtained, indexing starting at 0</td>
</tr>
</tbody>
</table>

Returns

The format of the specified output

7.15.22.14 aclmdlGetInputDataType

Description

Obtains the data type of an input in a model.

Prototype

```
aclDataType aclmdlGetInputDataType(const aclmdlDesc *modelDesc, size_t index)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelDesc</td>
<td>Input</td>
<td>Pointer to the data of the <code>aclmdlDesc</code> type</td>
</tr>
<tr>
<td>index</td>
<td>Input</td>
<td>Sequence number of the input whose data type is to be obtained, indexing starting at 0</td>
</tr>
</tbody>
</table>

Returns

The data type of the specified input
7.15.22.15 aclmdlGetOutputDataType

Description

Obtains the data type of an output in a model.

Prototype

```c
aclDataType aclmdlGetOutputDataType(const aclmdlDesc *modelDesc, size_t index)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelDesc</td>
<td>Input</td>
<td>Pointer to the data of the <code>aclmdlDesc</code> type</td>
</tr>
<tr>
<td>index</td>
<td>Input</td>
<td>Sequence number of the output whose data type is to be obtained, indexing starting at 0</td>
</tr>
</tbody>
</table>

Returns

The data type of the specified output

7.15.22.16 aclmdlGetInputIndexByName

Description

Obtains the index of an input using the input name in a model.

Prototype

```c
aclError aclmdlGetInputIndexByName(const aclmdlDesc *modelDesc, const char *name, size_t *index)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelDesc</td>
<td>Input</td>
<td>Pointer to the data of the <code>aclmdlDesc</code> type</td>
</tr>
<tr>
<td>name</td>
<td>Input</td>
<td>Input name</td>
</tr>
<tr>
<td>index</td>
<td>Output</td>
<td>Index of the input to be obtained</td>
</tr>
</tbody>
</table>
Returns

0 indicates success. Other values indicate failure.

7.15.22.17 aclmdlGetOutputIndexByName

Description

Obtains the index of the output by the output name in the model.

Prototype

aclError aclmdlGetOutputIndexByName(const aclmdlDesc *modelDesc, const char *name, size_t *index)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelDesc</td>
<td>Input</td>
<td>Pointer to the data of the aclmdlDesc type</td>
</tr>
<tr>
<td>name</td>
<td>Input</td>
<td>Output name</td>
</tr>
<tr>
<td>index</td>
<td>Output</td>
<td>Index of the output to be obtained</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.15.22.18 aclmdlGetDynamicBatch

Description

Obtains dynamic batch information supported by the model.

Prototype

aclError aclmdlGetDynamicBatch(const aclmdlDesc *modelDesc, aclmdlBatch *batch)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelDesc</td>
<td>Input</td>
<td>Pointer to the data of the aclmdlDesc type</td>
</tr>
</tbody>
</table>
### Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>batch</td>
<td>Output</td>
<td>const int ACL_MAX_BATCH_NUM = 128; typedef struct aclmdlBatch { size_t batchCount; /<strong>&lt; number of batch choices supported by the model*/ uint64_t batch[ACL_MAX_BATCH_NUM]; /</strong>&lt; specific batch choices supported by the model*/ } aclmdlBatch; If <strong>batchCount</strong> is 0, dynamic batch size is not supported. The batch size of the model applies.</td>
</tr>
</tbody>
</table>

### Returns

0 indicates success. Other values indicate failure.

### 7.15.22.19 aclmdlGetDynamicHW

**Description**

Obtains the dynamic size information supported by a model.

**Prototype**

```c
aclError aclmdlGetDynamicHW(const aclmdlDesc *modelDesc, size_t index, aclmdlHW *hw)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelDesc</td>
<td>Input</td>
<td>Pointer to the data of the <strong>aclmdlDesc</strong> type</td>
</tr>
<tr>
<td>index</td>
<td>Input</td>
<td>Reserved. The value is fixed to –1.</td>
</tr>
<tr>
<td>hw</td>
<td>Output</td>
<td>const int ACL_MAX_HW_NUM = 128; typedef struct aclmdlHW { size_t hwCount; /<strong>&lt; number of image size choices supported by the model*/ uint64_t hw[ACL_MAX_HW_NUM][2]; /</strong>&lt; Supported dynamic image size choices. In each array, element 0 indicates the height, and element 1 indicates the width.*/ } aclmdlHW; If <strong>hwCount</strong> is 0, dynamic image size is not supported. The image size of the model applies.</td>
</tr>
</tbody>
</table>

### Returns

0 indicates success. Other values indicate failure.
7.15.22.20 aclmdlGetCurOutputDims

Description

Obtains actual dimension information of the output tensor based on the model description.

In the dynamic shape scenario, if you have called `aclmdlSetDynamicBatchSize` to set the batch size choices or called `aclmdlSetDynamicHWSize` to set the image size choices, you can call this API to obtain the actual dimension information of the output tensor of the specified model. If `aclmdlSetDynamicBatchSize` or `aclmdlSetDynamicHWSize` is not called, this API can be used to obtain the dimension information of the maximum batch/image size choice.

In the fixed shape scenario, this API is used to obtain the dimension information of the output tensor of a model.

Prototype

```c
aclError aclmdlGetCurOutputDims(const aclmdlDesc *modelDesc, size_t index, aclmdlIODims *dims)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelDesc</td>
<td>Input</td>
<td>Pointer to the data of the <code>aclmdlDesc</code> type</td>
</tr>
<tr>
<td>index</td>
<td>Input</td>
<td>Sequence number of the output whose <code>dims</code> information is to be obtained, indexed starting at 0</td>
</tr>
<tr>
<td>dims</td>
<td>Output</td>
<td>Actual dimension information</td>
</tr>
</tbody>
</table>

```c
const int ACL_MAX_DIM_CNT = 128;
const int ACL_MAX_TENSOR_NAME_LEN = 128;
typedef struct aclmdlIODims {
    char name[ACL_MAX_TENSOR_NAME_LEN]; /**< tensor name */
    size_t dimCount; /**< Dimension count */
    int64_t dims[ACL_MAX_DIM_CNT]; /**< Dimension information */
} aclmdlIODims;
```

Returns

`0` indicates success. Other values indicate failure.

7.15.22.21 aclmdlGetFirstAippInfo

Description

Obtains the static AIPP configuration of a model.
Static AIPP provides the following functions. They are executed in the exact sequence. For details, see ATC Tool Instructions.

Cropping -> CSC -> mean subtraction/normalization -> Padding

**Prototype**

```c
aclError aclmdlGetFirstAippInfo(uint32_t modelId, size_t index, aclAippInfo *aippinfo)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modelId</td>
<td>Input</td>
<td>Model ID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A successful <code>aclmdlLoadFromFile</code>, <code>aclmdlLoadFromMem</code>, <code>aclmdlLoadFromFileWithMem</code>, or <code>aclmdlLoadFromMemWithMem</code> call returns a model ID.</td>
</tr>
<tr>
<td>index</td>
<td>Input</td>
<td>Sequence number of the specified input to the model</td>
</tr>
<tr>
<td>aippinfo</td>
<td>Output</td>
<td>Static AIPP configuration</td>
</tr>
</tbody>
</table>

**Returns**

0 indicates success. Other values indicate failure.

### 7.15.23 aclTensorDesc

The following interfaces can be used to perform relevant operations on data of the `aclTensorDesc` type. Interfaces of this data type are all synchronous interfaces.

#### 7.15.23.1 aclCreateTensorDesc

**Description**

Creates data of the `aclTensorDesc` type.

**Prototype**

```c
aclTensorDesc *aclCreateTensorDesc(aclDataType dataType,
int numDims,
const int64_t *dims,
aclFormat format)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataType</td>
<td>Input</td>
<td>Data type of the tensor description</td>
</tr>
<tr>
<td>numDims</td>
<td>Input</td>
<td>Number of shape dimensions of the tensor description</td>
</tr>
<tr>
<td></td>
<td></td>
<td>numDims has the following restrictions:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● numDims &gt;= 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● If numDims &gt; 0, then numDims must be the same as the length of the dims array.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● If numDims = 0, the dims array is ignored.</td>
</tr>
<tr>
<td>dims</td>
<td>Input</td>
<td>Size of a specified dimension of the tensor description</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dims is an array. Each element in the array indicates the size of each dimension in the tensor. If the value of an element in the array is 0, the tensor is empty.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If an empty tensor is needed, the minimum value to be set is 1 byte.</td>
</tr>
<tr>
<td>format</td>
<td>Input</td>
<td>Format of the tensor description</td>
</tr>
</tbody>
</table>

Returns

aclTensorDesc address

7.15.23.2 aclDestroyTensorDesc

Description

Destroys data of the aclTensorDesc type.

Prototype

void aclDestroyTensorDesc(const aclTensorDesc *desc)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>desc</td>
<td>Input</td>
<td>Pointer to the aclTensorDesc to be destroyed</td>
</tr>
</tbody>
</table>
7.15.23.3 aclGetTensorDescType

Description

Obtains the data type of a tensor description.

Prototype

```
aclDataType aclGetTensorDescType(const aclTensorDesc *desc)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>desc</td>
<td>Input</td>
<td>Pointer to the aclTensorDesc</td>
</tr>
</tbody>
</table>

Returns

Data type of the tensor description

7.15.23.4 aclGetTensorDescFormat

Description

Obtains the format of a tensor description.

Prototype

```
aclFormat aclGetTensorDescFormat(const aclTensorDesc *desc)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>desc</td>
<td>Input</td>
<td>Pointer to the aclTensorDesc</td>
</tr>
</tbody>
</table>

Returns

Format of the tensor description

7.15.23.5 aclGetTensorDescSize

Description

Obtains the size of a tensor description.
Prototype

```c
size_t aclGetTensorDescSize(const aclTensorDesc *desc)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>desc</td>
<td>Input</td>
<td>Pointer to the aclTensorDesc</td>
</tr>
</tbody>
</table>

Returns

Size of the tensor description

7.15.23.6 aclGetTensorDescElementCount

Description

Obtains the number of elements of a tensor description.

Prototype

```c
size_t aclGetTensorDescElementCount(const aclTensorDesc *desc)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>desc</td>
<td>Input</td>
<td>Pointer to the aclTensorDesc</td>
</tr>
</tbody>
</table>

Returns

Number of elements of the tensor description

7.15.23.7 aclGetTensorDescNumDims

Description

Obtains the number of shape dimensions of a tensor description.

Prototype

```c
size_t aclGetTensorDescNumDims(const aclTensorDesc *desc)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>desc</td>
<td>Input</td>
<td>Pointer to the aclTensorDesc</td>
</tr>
</tbody>
</table>

Returns

Dimensions number of the shape of the tensor description

7.15.23.8 aclGetTensorDescDim

Description

Obtains the size of a specified dimension of a tensor description.

Prototype

```c
int64_t aclGetTensorDescDim(const aclTensorDesc *desc, size_t index)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>desc</td>
<td>Input</td>
<td>Pointer to the aclTensorDesc</td>
</tr>
<tr>
<td>index</td>
<td>Input</td>
<td>Sequence number of the dimension whose size is to be obtained. The index value starts from 0.</td>
</tr>
</tbody>
</table>

Returns

Size of the specified dimension of the tensor description

7.15.23.9 aclSetTensorDescName

Description

Sets the TensorDesc name.

Prototype

```c
void aclSetTensorDescName(aclTensorDesc *desc, const char *name)
```
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>desc</td>
<td>Input</td>
<td>Pointer to the <code>aclTensorDesc</code></td>
</tr>
<tr>
<td>name</td>
<td>Input</td>
<td><code>TensorDesc</code> name</td>
</tr>
</tbody>
</table>

### Returns

None

#### 7.15.23.10 aclGetTensorDescName

**Description**

Obtains the `TensorDesc` name.

**Prototype**

```c
const char *aclGetTensorDescName(aclTensorDesc *desc)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>desc</td>
<td>Input</td>
<td>Pointer to the <code>aclTensorDesc</code></td>
</tr>
</tbody>
</table>

**Returns**

Name of the `TensorDesc`

#### 7.15.23.11 aclTransTensorDescFormat

**Description**

(Reserved) Creates `aclTensorDesc` with the `dstFormat` format from the source `aclTensorDesc`.

**Prototype**

```c
aclError aclTransTensorDescFormat(const aclTensorDesc *srcDesc, aclFormat dstFormat, aclTensorDesc **dstDesc)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>srcDesc</td>
<td>Input</td>
<td>Pointer to source aclTensorDesc data</td>
</tr>
<tr>
<td>dstFormat</td>
<td>Input</td>
<td>Target format</td>
</tr>
<tr>
<td>dstDesc</td>
<td>Output</td>
<td>Pointer to the pointer to the target aclTensorDesc data</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.15.23.12 aclSetTensorStorageShape

Description

Sets the actual dims of the tensor after creating the tensor description by calling aclCreateTensorDesc. This API is mainly used in the PyTorch scenario.

Prototype

aclError aclSetTensorStorageShape(aclTensorDesc *desc, int numDims, const int64_t *dims)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>desc</td>
<td>Input</td>
<td>Pointer to the aclTensorDesc data</td>
</tr>
<tr>
<td>numDims</td>
<td>Input</td>
<td>Number of dimensions</td>
</tr>
<tr>
<td>dims</td>
<td>Input</td>
<td>Dimensions</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.15.23.13 aclSetTensorStorageFormat

Description

Sets the actual format of the tensor after creating the tensor description by calling aclCreateTensorDesc. This API is mainly used in the PyTorch scenario.
Prototype

\texttt{aclError aclSetTensorStorageFormat(aclTensorDesc *desc, aclFormat format)}

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>desc</td>
<td>Input</td>
<td>Pointer to the \texttt{aclTensorDesc} data</td>
</tr>
<tr>
<td>format</td>
<td>Input</td>
<td>Target format</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.15.24 aclopAttr

The following interfaces are used to save the attributes of an operator and set data of the \texttt{aclopAttr} type. Interfaces of this data type are all synchronous interfaces.

7.15.24.1 aclopCreateAttr

Description

Creates data of the \texttt{aclopAttr} type.

Prototype

\texttt{aclopAttr *aclopCreateAttr()}

Parameters

None

Returns

\texttt{aclopAttr address}

7.15.24.2 aclopDestroyAttr

Description

Destroys data of the \texttt{aclopAttr} type.

Prototype

\texttt{void aclopDestroyAttr(const aclopAttr *attr)}
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr</td>
<td>Input</td>
<td>Pointer to the <code>aclopAttr</code> to be destroyed</td>
</tr>
</tbody>
</table>

Returns

None

7.15.24.3 aclopSetAttrBool

Description

Sets the attribute value of a scalar of the bool type.

Prototype

```c
aclError aclopSetAttrBool(aclopAttr *attr, const char *attrName, uint8_t attrValue)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr</td>
<td>Input</td>
<td>Pointer to the <code>aclopAttr</code></td>
</tr>
<tr>
<td>attrName</td>
<td>Input</td>
<td>Attribute name</td>
</tr>
<tr>
<td>attrValue</td>
<td>Input</td>
<td>Attribute value</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.15.24.4 aclopSetAttrInt

Description

Sets the attribute value of a scalar of the int64_t type.

Prototype

```c
aclError aclopSetAttrInt(aclopAttr *attr, const char *attrName, int64_t attrValue);
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr</td>
<td>Input</td>
<td>Pointer to the aclopAttr</td>
</tr>
<tr>
<td>attrName</td>
<td>Input</td>
<td>Attribute name</td>
</tr>
<tr>
<td>attrValue</td>
<td>Input</td>
<td>Attribute value</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.15.24.5 aclopSetAttrFloat

Description

Sets the attribute value of a scalar of the float type.

Prototype

aclError aclopSetAttrFloat(aclopAttr *attr, const char *attrName, float attrValue)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr</td>
<td>Input</td>
<td>Pointer to the aclopAttr</td>
</tr>
<tr>
<td>attrName</td>
<td>Input</td>
<td>Attribute name</td>
</tr>
<tr>
<td>attrValue</td>
<td>Input</td>
<td>Attribute value</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.15.24.6 aclopSetAttrString

Description

Sets the attribute value of a scalar of the string type.

Prototype

aclError aclopSetAttrString(aclopAttr *attr, const char *attrName, const char *attrValue)
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr</td>
<td>Input</td>
<td>Pointer to the <code>aclopAttr</code></td>
</tr>
<tr>
<td>attrName</td>
<td>Input</td>
<td>Attribute name</td>
</tr>
<tr>
<td>attrValue</td>
<td>Input</td>
<td>Attribute value</td>
</tr>
</tbody>
</table>

### Returns

0 indicates success. Other values indicate failure.

#### 7.15.24.7 aclopSetAttrListBool

### Description

Sets the attribute value of a list of bools.

### Prototype

```c
aclError aclopSetAttrListBool(aclopAttr *attr, const char *attrName, int numValues, const uint8_t *values)
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr</td>
<td>Input</td>
<td>Pointer to the <code>aclopAttr</code></td>
</tr>
<tr>
<td>attrName</td>
<td>Input</td>
<td>Attribute name</td>
</tr>
<tr>
<td>numValues</td>
<td>Input</td>
<td>Number of attribute values</td>
</tr>
<tr>
<td>values</td>
<td>Input</td>
<td>Array address of the attribute values</td>
</tr>
</tbody>
</table>

### Returns

0 indicates success. Other values indicate failure.

#### 7.15.24.8 aclopSetAttrListInt

### Description

Sets the attribute value of a list of int64_ts.
Prototype

```c
aclError aclopSetAttrListInt(aclopAttr *attr, const char *attrName, int numValues, const int64_t *values)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr</td>
<td>Input</td>
<td>Pointer to the <code>aclopAttr</code></td>
</tr>
<tr>
<td>attrName</td>
<td>Input</td>
<td>Attribute name</td>
</tr>
<tr>
<td>numValues</td>
<td>Input</td>
<td>Number of attribute values</td>
</tr>
<tr>
<td>values</td>
<td>Input</td>
<td>Array address of the attribute values</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.15.24.9 `aclopSetAttrListFloat`

Description

Sets the attribute value of a list of floats.

Prototype

```c
aclError aclopSetAttrListFloat(aclopAttr *attr, const char *attrName, int numValues, const float *values)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr</td>
<td>Input</td>
<td>Pointer to the <code>aclopAttr</code></td>
</tr>
<tr>
<td>attrName</td>
<td>Input</td>
<td>Attribute name</td>
</tr>
<tr>
<td>numValues</td>
<td>Input</td>
<td>Number of attribute values</td>
</tr>
<tr>
<td>values</td>
<td>Input</td>
<td>Array address of the attribute values</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.
7.15.24.10 aclopSetAttrListString

Description
Sets the attribute value of a list of strings.

Prototype

```c
aclError aclopSetAttrListString(aclopAttr *attr, const char *attrName, int numValues, const char **values)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr</td>
<td>Input</td>
<td>Pointer to the <code>aclopAttr</code></td>
</tr>
<tr>
<td>attrName</td>
<td>Input</td>
<td>Attribute name</td>
</tr>
<tr>
<td>numValues</td>
<td>Input</td>
<td>Number of attribute values</td>
</tr>
<tr>
<td>values</td>
<td>Input</td>
<td>Pointer to the pointer to the array attribute values</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.15.24.11 aclopSetAttrListListInt

Description
Sets the attribute values of lists of int64_t.

Prototype

```c
aclError aclopSetAttrListListInt(aclopAttr *attr, const char *attrName, int numLists, const int *numValues, const int64_t *const values[]);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr</td>
<td>Input</td>
<td>Pointer to the <code>aclopAttr</code></td>
</tr>
<tr>
<td>attrName</td>
<td>Input</td>
<td>Attribute names</td>
</tr>
</tbody>
</table>

Issue 01 (2020-07-30) Copyright © Huawei Technologies Co., Ltd. 301
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numLists</td>
<td>Input</td>
<td>Number of lists</td>
</tr>
<tr>
<td>numValues</td>
<td>Input</td>
<td>Number of attribute values in each list</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Each element in the numValues array indicates the number of attribute values in each list.</td>
</tr>
<tr>
<td>values</td>
<td>Input</td>
<td>Attribute values in each list</td>
</tr>
</tbody>
</table>

**Returns**

0 indicates success. Other values indicate failure.

### 7.15.25 acldvppChannelDesc

The following interfaces are used to describe the channel information about image data processing, as well as create and destroy data of the acldvppChannelDesc type. Interfaces of this data type are all synchronous interfaces.

#### 7.15.25.1 acldvppCreateChannelDesc

**Description**

Creates data of the acldvppChannelDesc type to describe an image data processing channel. It is a synchronous interface.

**Prototype**

```c
acldvppChannelDesc *acldvppCreateChannelDesc()
```

**Parameters**

None

**Returns**

- **Success**: acldvppChannelDesc type
- **Failure**: null

#### 7.15.25.2 acldvppDestroyChannelDesc

**Description**

Destroys data of the acldvppChannelDesc type created by calling acldvppCreateChannelDesc. It is a synchronous interface.
Before calling `acldvppDestroyChannelDesc` to destroy data of the `acldvppChannelDesc` type, `acldvppDestroyChannel` must be called to destroy the channel. Otherwise, an error is reported.

**Prototype**

```
acLError acldvppDestroyChannelDesc(acldvppChannelDesc *channelDesc)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelDesc</td>
<td>Input</td>
<td><code>channelDesc</code> to be destroyed</td>
</tr>
</tbody>
</table>

**Returns**

0 indicates success. Other values indicate failure.

### 7.15.25.3 acldvppGetChannelDescChannelId

**Description**

Obtains channel ID with the channel description.

**Restrictions**

Interfaces calling sequence: `acldvppCreateChannelDesc` -> `acldvppCreateChannel` -> `acldvppGetChannelDescChannelId`

**Prototype**

```
uint64_t acldvppGetChannelDescChannelId(const acldvppChannelDesc *channelDesc)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelDesc</td>
<td>Input</td>
<td>Channel description</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data of the <code>acldvppChannelDesc</code> type is created by calling <code>acldvppCreateChannelDesc</code>.</td>
</tr>
</tbody>
</table>

**Returns**

Channel ID
### 7.15.25.4 acldvppSetChannelDescMode

**Description**

Specifies the channel mode in the channel description, and specifies the function to be implemented by the image data processing channel. Currently, this interface supports **VPC**, **JPEGD** and **JPEGE**.

You are advised to specify the channel mode as required using this interface. Otherwise, channels in the three modes are created by default, which may occupy resources.

The current version does not support this interface.

**Restrictions**

Interface calling sequence: 

```
acldvppCreateChannelDesc->
acldvppSetChannelDescMode-> acldvppCreateChannel
```

**Prototype**

```c
aclError acldvppSetChannelDescMode(acldvppChannelDesc *channelDesc, uint32_t mode)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelDesc</td>
<td>Input</td>
<td>Channel description</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data of the <strong>acldvppChannelDesc</strong> type is created by calling</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>acldvppCreateChannelDesc</strong>.</td>
</tr>
<tr>
<td>mode</td>
<td>Input</td>
<td>Channel mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>enum acldvppChannelMode {</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>DVPP_CHNMODE_VPC = 1,</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>DVPP_CHNMODE_JPEGD = 2,</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>DVPP_CHNMODE_JPEGE = 4</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>};</code></td>
</tr>
</tbody>
</table>

**Returns**

0 indicates success. Other values indicate failure.

### 7.15.26 acldvppPicDesc

Interfaces of this data type are all synchronous interfaces.

### 7.15.26.1 acldvppCreatePicDesc

**Description**

Creates image description.
Prototype

\[ \text{aclvppPicDesc} * \text{aclvppCreatePicDesc}() \]

Parameters

None

Returns

- Success: \text{aclvppPicDesc} type
- Failure: \text{null}

7.15.26.2 aclvppSetPicDesc Series

Description

Sets image description parameters.

Prototype

\[ \text{aclError} \text{ aclvppSetPicDescData(aclvppPicDesc} * \text{picDesc, void} \ast \text{dataDev}); \]
\[ \text{aclError} \text{ aclvppSetPicDescSize(aclvppPicDesc} * \text{picDesc, uint32_t size}); \]
\[ \text{aclError} \text{ aclvppSetPicDescFormat(aclvppPicDesc} * \text{picDesc, aclvppPixelFormat format}); \]
\[ \text{aclError} \text{ aclvppSetPicDescWidth(aclvppPicDesc} * \text{picDesc, uint32_t width}); \]
\[ \text{aclError} \text{ aclvppSetPicDescHeight(aclvppPicDesc} * \text{picDesc, uint32_t height}); \]
\[ \text{aclError} \text{ aclvppSetPicDescWidthStride(aclvppPicDesc} * \text{picDesc, uint32_t widthStride}); \]
\[ \text{aclError} \text{ aclvppSetPicDescHeightStride(aclvppPicDesc} * \text{picDesc, uint32_t heightStride}); \]
\[ \text{aclError} \text{ aclvppSetPicDescRetCode(aclvppPicDesc} * \text{picDesc, uint32_t retCode}) \]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>picDesc</td>
<td>Input</td>
<td>Image description</td>
</tr>
<tr>
<td>dataDev</td>
<td>Input</td>
<td>Image data memory on the device, allocated by using aclvppMalloc</td>
</tr>
<tr>
<td>size</td>
<td>Input</td>
<td>Memory size, in bytes</td>
</tr>
<tr>
<td>format</td>
<td>Input</td>
<td>Image format</td>
</tr>
<tr>
<td>width</td>
<td>Input</td>
<td>Width of the source image</td>
</tr>
<tr>
<td>height</td>
<td>Input</td>
<td>Height of the source image</td>
</tr>
<tr>
<td>Parameter</td>
<td>Input/Output</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------</td>
<td>------------------------------------------------------------</td>
</tr>
<tr>
<td>widthStride</td>
<td>Input</td>
<td>Width after alignment</td>
</tr>
<tr>
<td>heightStride</td>
<td>Input</td>
<td>Height after alignment</td>
</tr>
<tr>
<td>retCode</td>
<td>Input</td>
<td>Return code (0: success; other values: failure)</td>
</tr>
</tbody>
</table>

**Returns**

0 indicates success. Other values indicate failure.

### 7.15.26.3 aclDvppGetPicDesc Series

**Description**

Obtains image description.

**Prototype**

```c
void *aclDvppGetPicDescData(const aclDvppPicDesc *picDesc);
uint32_t aclDvppGetPicDescSize(const aclDvppPicDesc *picDesc);
acldvppPixelFormat aclDvppGetPicDescFormat(const aclDvppPicDesc *picDesc);
uint32_t aclDvppGetPicDescWidth(const aclDvppPicDesc *picDesc);
uint32_t aclDvppGetPicDescHeight(const aclDvppPicDesc *picDesc);
uint32_t aclDvppGetPicDescWidthStride(const aclDvppPicDesc *picDesc);
uint32_t aclDvppGetPicDescHeightStride(const aclDvppPicDesc *picDesc);
uint32_t aclDvppGetPicDescRetCode(const aclDvppPicDesc *picDesc);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>picDesc</td>
<td>Input</td>
<td>Image description</td>
</tr>
</tbody>
</table>

**Returns**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Memory address of the image data</td>
</tr>
<tr>
<td>size</td>
<td>Memory size of the image data</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>format</td>
<td>Image format</td>
</tr>
<tr>
<td>width</td>
<td>Width of the original image</td>
</tr>
<tr>
<td>height</td>
<td>Height of the original image</td>
</tr>
<tr>
<td>widthStride</td>
<td>Width after alignment</td>
</tr>
<tr>
<td>heightStride</td>
<td>Height after alignment</td>
</tr>
<tr>
<td>retCode</td>
<td>Return code (0: success; other values: failure)</td>
</tr>
</tbody>
</table>

**7.15.26.4 acldvppDestroyPicDesc**

**Description**

Destroys image description created by calling `acldvppCreatePicDesc`.

**Prototype**

```c
aclError acldvppDestroyPicDesc(acldvppPicDesc *picDesc)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>picDesc</td>
<td>Input</td>
<td>Image description to be destroyed</td>
</tr>
</tbody>
</table>

**Returns**

0 indicates success. Other values indicate failure.

**7.15.27 acldvppRoiConfig**

Interfaces of this data type are all synchronous interfaces.

**7.15.27.1 acldvppCreateRoiConfig**

**Description**

Creates data to describe the location of a region of interest (ROI).

**Prototype**

```c
acldvppRoiConfig *acldvppCreateRoiConfig(
    uint32_t left,
    uint32_t right,
```
uint32_t top,
uint32_t bottom)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
</table>
| left      | Input        | Left offset
Must be an even number.
If the data is used to describe the pasteArea, the left-offset must be rounded up to the nearest multiple of 16. |
| right     | Input        | Right offset
Must be an odd number. |
| top       | Input        | Top offset
Must be an even number. |
| bottom    | Input        | Bottom offset
Must be an odd number. |

Returns

- Success: aclDvppRoiConfig type
- Failure: null

7.15.27.2 aclDvppSetRoiConfig Series

Description

Sets the location of an ROI.

Prototype

```c
aclError aclDvppSetRoiConfig(aclDvppRoiConfig *config, uint32_t left, uint32_t right, uint32_t top, uint32_t bottom);

aclError aclDvppSetRoiConfigLeft (aclDvppRoiConfig *config, uint32_t left);

aclError aclDvppSetRoiConfigRight (aclDvppRoiConfig *config, uint32_t right);

aclError aclDvppSetRoiConfigTop (aclDvppRoiConfig *config, uint32_t top);

aclError aclDvppSetRoiConfigBottom(aclDvppRoiConfig *config, uint32_t bottom)
```
# Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
</table>
| config | Input | Location data of the ROI
Create data of the acldvppRoiConfig type by calling acldvppCreateRoiConfig. |
| left | Input | Left offset
Must be an even number.
If the data is used to describe the pasteArea, the left-offset must be rounded up to the nearest multiple of 16. |
| right | Input | Right offset
Must be an odd number. |
| top | Input | Top offset
Must be an even number. |
| bottom | Input | Bottom offset
Must be an odd number. |

## Returns

0 indicates success. Other values indicate failure.

### 7.15.27.3 acldvppDestroyRoiConfig

#### Description

Destroys data created by calling acldvppCreateRoiConfig.

#### Prototype

```c
aclError acldvppDestroyRoiConfig(acldvppRoiConfig *roiConfig)
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>roiConfig</td>
<td>Input</td>
<td>Data to be destroyed</td>
</tr>
</tbody>
</table>

## Returns

0 indicates success. Other values indicate failure.
7.15.28 aclDvppResizeConfig

Interfaces of this data type are all synchronous interfaces.

7.15.28.1 aclDvppCreateResizeConfig

Description

Creates image resizing configuration data. The nearest neighbor interpolation algorithm developed by Huawei is used by default.

Prototype

aclDvppResizeConfig *aclDvppCreateResizeConfig()

Parameters

None

Returns

- Success: aclDvppResizeConfig type
- Failure: null

7.15.28.2 aclDvppSetResizeConfigInterpolation

Description

Sets the image resizing algorithm.

Prototype

aclError aclDvppSetResizeConfigInterpolation(aclDvppResizeConfig *resizeConfig, uint32_t interpolation)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resizeConfig</td>
<td>Input</td>
<td>Resizing configuration data to be set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Create data of the aclDvppResizeConfig type by calling aclDvppCreateResizeConfig.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Input/Output</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>interpolation</td>
<td>Input</td>
<td>Resizing operator select. The values are as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 0 (default value): Huawei-developed nearest neighbor interpolation algorithm</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 1: The frequently-used bilinear algorithm (not supported currently)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 2: The frequently-used nearest neighbor algorithm (not supported currently)</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.15.28.3 acldevppGetResizeConfigInterpolation

Description

Obtains the image resizing algorithm.

Prototype

uint32 acldevppGetResizeConfigInterpolation(const acldevppResizeConfig * resizeConfig)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resizeConfig</td>
<td>Input</td>
<td>Resizing configuration data to be obtained</td>
</tr>
</tbody>
</table>

Returns

● 0 (default): Huawei-developed nearest neighbor interpolation algorithm

7.15.28.4 acldevppDestroyResizeConfig

Description

Destroys resizing configuration data created by calling acldevppCreateResizeConfig.
Prototype

```
aclvppDestroyResizeConfig(aclvppResizeConfig *resizeConfig)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resizeConfig</td>
<td>Input</td>
<td>Data to be destroyed</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.15.29 aclvppJpegeConfig

 Interfaces of this data type are all synchronous interfaces.

7.15.29.1 aclvppCreateJpegeConfig

Description

Creates image encoding configuration data.

Prototype

```
aclvppJpegeConfig *aclvppCreateJpegeConfig()
```

Parameters

None

Returns

- Success: aclvppResizeConfig type
- Failure: null

7.15.29.2 aclvppSetJpegeConfigLevel

Description

Sets encoding configuration data.

Prototype

```
aclvppSetJpegeConfigLevel(aclvppJpegeConfig *jpegeConfig, uint32_t level)
```
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jpegeConfig</td>
<td>Input/Output</td>
<td>Encoding configuration data Create data of the acldvppJpegeConfig type by calling acldvppCreateJpegeConfig.</td>
</tr>
<tr>
<td>level</td>
<td>Input</td>
<td>Encoding quality in a range of [0, 100]. The encoding quality of level 0 is similar to that of level 100. For the value range of [1, 100], a smaller value indicates poorer output image quality.</td>
</tr>
</tbody>
</table>

### Returns

`0` indicates success. Other values indicate failure.

### 7.15.29.3 aclvppGetJpegeConfigLevel

#### Description

Obtains encoding configuration.

#### Prototype

`uint32_t aclvppGetJpegeConfigLevel(const aclvppJpegeConfig *jpegeConfig)`

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jpegeConfig</td>
<td>Input</td>
<td>Encoding configuration data Create data of the acldvppJpegeConfig type by calling acldvppCreateJpegeConfig, and set the encoding configuration by calling aclvppSetJpegeConfigLevel.</td>
</tr>
</tbody>
</table>

### Returns

Encoding quality in the range of [0, 100]. The encoding quality of level 0 is similar to that of level 100. For the value range of [1, 100], a smaller value indicates poorer output image quality.
7.15.29.4 acldvppDestroyJpegeConfig

Description

Destroys encoding configuration data created by calling acldvppCreateJpegeConfig.

Prototype

aclError acldvppDestroyJpegeConfig(acldvppJpegeConfig *jpegeConfig)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jpegeConfig</td>
<td>Input</td>
<td>Data to be destroyed</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.15.30 aclvdecChannelDesc

Interfaces of this data type are all synchronous interfaces.

7.15.30.1 aclvdecCreateChannelDesc

Description

Creates data of the aclvdecChannelDesc type to describe a video decoding channel. It is a synchronous interface.

Prototype

aclvdecChannelDesc *aclvdecCreateChannelDesc()

Parameters

None

Returns

- Success: aclvdecChannelDesc type
- Failure: null

7.15.30.2 aclvdecSetChannelDesc Series Interfaces

Description

Sets the description of a video decoding channel.
Prototype

```c
aclError aclvdecSetChannelDescChannelId(aclvdecChannelDesc *channelDesc, uint32_t channelId);

aclError aclvdecSetChannelDescThreadId(aclvdecChannelDesc *channelDesc, uint64_t threadId);

aclError aclvdecSetChannelDescCallback(aclvdecChannelDesc *channelDesc, aclvdecCallback callback);

aclError aclvdecSetChannelDescEnType(aclvdecChannelDesc *channelDesc, acldvppStreamFormat enType);

aclError aclvdecSetChannelDescOutPicFormat(aclvdecChannelDesc *channelDesc, acldvppPixelFormat outPicFormat);

aclError aclvdecSetChannelDescOutPicWidth(aclvdecChannelDesc *channelDesc, uint32_t outPicWidth);

aclError aclvdecSetChannelDescOutPicHeight(aclvdecChannelDesc *channelDesc, uint32_t outPicHeight);

aclError aclvdecSetChannelDescRefFrameNum(aclvdecChannelDesc *channelDesc, uint32_t refFrameNum);

aclError aclvdecSetChannelDescOutMode(aclvdecChannelDesc *channelDesc, uint32_t outMode)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelDesc</td>
<td>Input</td>
<td>Description of the video decoding channel</td>
</tr>
<tr>
<td>channelId</td>
<td>Input</td>
<td>Decoding channel ID. The value range is [0, 31].</td>
</tr>
<tr>
<td>threadId</td>
<td>Input</td>
<td>Callback thread ID</td>
</tr>
<tr>
<td>callback</td>
<td>Input</td>
<td>Decoding callback function</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For details, see 7.12.7.3 aclvdecCallback.</td>
</tr>
<tr>
<td>enType</td>
<td>Input</td>
<td>Video encoding protocol: H.265 main profile (0), H.264 baseline profile (1), H.264 main profile (2), or H.264 high profile (3)</td>
</tr>
<tr>
<td>outPicFormat</td>
<td>Input</td>
<td>YUV image format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● YUV420SP NV12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● YUV420SP NV21</td>
</tr>
<tr>
<td>outPicWidth</td>
<td>Input</td>
<td>Image width</td>
</tr>
</tbody>
</table>
### Parameter Table

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outPicHeight</td>
<td>Input</td>
<td>Image height</td>
</tr>
<tr>
<td>refFrameNum</td>
<td>Input</td>
<td>Number of reference frames. The value range is [0, 16].</td>
</tr>
<tr>
<td>outMode</td>
<td>Input</td>
<td>Frame output mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 0: delayed frame output mode. The VDEC starts to output the decoding result only after receiving multiple frames in the stream.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● 1: real-time frame output mode. The VDEC starts to output the decoding result in real time after obtaining a frame in the stream. Only H.264/H.265 standard streams with simple reference relationships are supported (no long-term reference frames or B-frames).</td>
</tr>
</tbody>
</table>

### Returns

0 indicates success. Other values indicate failure.

### 7.15.3 aclvdecGetChannelDesc Series Interfaces

#### Description

Obtains the description of a video decoding channel.

#### Prototype

```c
uint32_t aclvdecGetChannelDescChannelId(const aclvdecChannelDesc *channelDesc);
uint32_t aclvdecGetChannelDescThreadId(const aclvdecChannelDesc *channelDesc);
aclvdecCallback aclvdecGetChannelDescCallback(const aclvdecChannelDesc *channelDesc);
acldvppStreamFormat aclvdecGetChannelDescEnType(const aclvdecChannelDesc *channelDesc);
acldvppPixelFormat aclvdecGetChannelDescOutPicFormat(const aclvdecChannelDesc *channelDesc);
uint32_t aclvdecGetChannelDescOutPicWidth(const aclvdecChannelDesc *channelDesc);
```
uint32_t aclvdecGetChannelDescOutPicHeight(const aclvdecChannelDesc *channelDesc);
uint32_t aclvdecGetChannelDescRefFrameNum(const aclvdecChannelDesc *channelDesc)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelDesc</td>
<td>Input</td>
<td>Description of the video decoding channel</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelId</td>
<td>Encoding channel ID. The value range is [0, 31].</td>
</tr>
<tr>
<td>threadId</td>
<td>Callback thread ID</td>
</tr>
<tr>
<td>callback</td>
<td>Callback function</td>
</tr>
<tr>
<td></td>
<td>For details, see 7.12.7.3 aclvdecCallback.</td>
</tr>
<tr>
<td>enType</td>
<td>Video encoding protocol: H.265 main profile (0),</td>
</tr>
<tr>
<td></td>
<td>H.264 baseline profile (1), H.264 main profile</td>
</tr>
<tr>
<td></td>
<td>(2), or H.264 high profile (3)</td>
</tr>
<tr>
<td>outPicFormat</td>
<td>YUV image storage format</td>
</tr>
<tr>
<td>outPicWidth</td>
<td>Image width</td>
</tr>
<tr>
<td>outPicHeight</td>
<td>Image height</td>
</tr>
<tr>
<td>refFrameNum</td>
<td>Number of reference frames. The value range is</td>
</tr>
<tr>
<td></td>
<td>[0, 16].</td>
</tr>
</tbody>
</table>

7.15.30.4 aclvdecDestroyChannelDesc

Description

Destroys data of the aclvdecChannelDesc type created by calling aclvdecCreateChannelDesc.

Prototype

aclError aclvdecDestroyChannelDesc(aclvdecChannelDesc *channelDesc)
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelDesc</td>
<td>Input</td>
<td><code>channelDesc</code> to be destroyed</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.15.31 acldvppStreamDesc

Interfaces of this data type are all synchronous interfaces.

7.15.31.1 acldvppCreateStreamDesc

Description

Creates data of the `acldvppStreamDesc` type to describe a video stream.

Prototype

```
acldvppStreamDesc *acldvppCreateStreamDesc()
```

Parameters

None

Returns

- Success: `acldvppStreamDesc` type
- Failure: `null`

7.15.31.2 acldvppSetStreamDesc Series Interfaces

Description

Sets the description of a video stream.

Prototype

```
acLError acldvppSetStreamDescData(acldvppStreamDesc *streamDesc, void *dataDev);
acLError acldvppSetStreamDescSize(acldvppStreamDesc *streamDesc, uint32_t size);
acLError acldvppSetStreamDescFormat(acldvppStreamDesc *streamDesc, acldvppStreamFormat format);
acLError acldvppSetStreamDescTimestamp(acldvppStreamDesc *streamDesc, uint64_t timestamp);
```
aclError aclDevpSetStreamDescRetCode(acldvppStreamDesc *streamDesc, uint32_t retCode);

aclError aclDevpSetStreamDescEos(acldvppStreamDesc *streamDesc, uint8_t eos)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>streamDesc</td>
<td>Input</td>
<td>Video stream description to be set</td>
</tr>
<tr>
<td>dataDev</td>
<td>Input</td>
<td>Stream data memory on the device</td>
</tr>
<tr>
<td>size</td>
<td>Input</td>
<td>Memory size, in bytes</td>
</tr>
<tr>
<td>format</td>
<td>Input</td>
<td>Stream format (H.264/H.265)</td>
</tr>
<tr>
<td>timestamp</td>
<td>Input</td>
<td>Timestamp</td>
</tr>
<tr>
<td>retCode</td>
<td>Input</td>
<td>Return code (0: success; other values: failure)</td>
</tr>
<tr>
<td>EOS</td>
<td>Input</td>
<td>EOS flag (0: no; 1: yes)</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.15.31.3 acldvppGetStreamDesc Series Interfaces

Description

Obtains the description of a video stream.

Prototype

void *acldvppGetStreamDescData(const acldvppStreamDesc *streamDesc);

uint32_t acldvppGetStreamDescSize(const acldvppStreamDesc *streamDesc);

acldvppStreamFormat acldvppGetStreamDescFormat(const acldvppStreamDesc *streamDesc);

uint64_t acldvppGetStreamDescTimestamp(const acldvppStreamDesc *streamDesc);

uint32_t acldvppGetStreamDescRetCode(const acldvppStreamDesc *streamDesc);

uint8_t acldvppGetStreamDescEos(const acldvppStreamDesc *streamDesc)
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>streamDesc</td>
<td>Input</td>
<td>Video stream description to be set</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataDev</td>
<td>Stream data memory on the device</td>
</tr>
<tr>
<td>size</td>
<td>Memory size, in bytes</td>
</tr>
<tr>
<td>format</td>
<td>Stream format (H.264/H.265)</td>
</tr>
<tr>
<td>timestamp</td>
<td>Timestamp</td>
</tr>
<tr>
<td>retCode</td>
<td>Return code (0: success; other values: failure)</td>
</tr>
<tr>
<td>EOS</td>
<td>EOS flag</td>
</tr>
</tbody>
</table>

7.15.31.4 acldvppDestroyStreamDesc

Description

Destroys data of the acldvppStreamDesc type created by calling acldvppCreateStreamDesc.

Prototype

```
aclError acldvppDestroyStreamDesc(acldvppStreamDesc *streamDesc)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>streamDesc</td>
<td>Input</td>
<td>streamDesc to be destroyed</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.15.32 aclvdecFrameConfig

Interfaces of this data type are all synchronous interfaces.
7.15.32.1 aclvdecCreateFrameConfig

Description

Creates data of the aclvdecFrameConfig type for the single-frame encoding configuration.

Prototype

aclvdecFrameConfig *aclvdecCreateFrameConfig()

Parameters

None

Returns

- Success: data of the aclvdecFrameConfig type
- Failure: null

7.15.32.2 aclvdecDestroyFrameConfig

Description

Destroys data of the aclvdecFrameConfig type created by calling aclvdecCreateFrameConfig.

Prototype

aclError aclvdecDestroyFrameConfig(aclvdecFrameConfig *vdecFrameConfig)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vdecFrameConfig</td>
<td>Input</td>
<td>vdecFrameConfig to be destroyed</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.15.33 acldvppBatchPicDesc

Interfaces of this data type are all synchronous interfaces.
7.15.33.1 aclvppCreateBatchPicDesc

Description

Creates data of the aclvppBatchPicDesc type, for the description of multiple images.

Prototype

aclvppBatchPicDesc *aclvppCreateBatchPicDesc(uint32_t batchSize)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>batchSize</td>
<td>Input</td>
<td>Image count</td>
</tr>
</tbody>
</table>

Returns

- Success: data of the aclvppBatchPicDesc type
- Failure: null

7.15.33.2 aclvppGetPicDesc

Description

Obtains the description of an image.

Prototype

aclvppPicDesc *aclvppGetPicDesc(aclvppBatchPicDesc *batchPicDesc, uint32_t index)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>batchPicDesc</td>
<td>Input</td>
<td>Description of multiple images</td>
</tr>
<tr>
<td>index</td>
<td>Input</td>
<td>Sequence number of the image whose description is to be obtained, indexed starting at 0</td>
</tr>
</tbody>
</table>

Returns

Description of the specified image
7.15.33.3 acldvppDestroyBatchPicDesc

Description

Destroys the data of the acldvppBatchPicDesc type created by calling acldvppCreateBatchPicDesc.

Prototype

aclError acldvppDestroyBatchPicDesc (acldvppBatchPicDesc *batchPicDesc)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>batchPicDesc</td>
<td>Input</td>
<td>acldvppBatchPicDesc data</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.15.34 aclvencChannelDesc

Interfaces of this data type are all synchronous interfaces.

7.15.34.1 aclvencCreateChannelDesc

Description

Creates data of the aclvencChannelDesc type to describe a video encoding channel.

Prototype

aclvencChannelDesc *aclvencCreateChannelDesc()

Parameters

None

Returns

- Success: data of the aclvencChannelDesc type
- Failure: null

7.15.34.2 aclvencSetChannelDesc series

Description

Sets the description of a video encoding channel.
Prototype

aclError aclvencSetChannelDescThreadId(aclvencChannelDesc *channelDesc, uint64_t threadId);
aclError aclvencSetChannelDescCallback(aclvencChannelDesc *channelDesc, aclvencCallback callback);
aclError aclvencSetChannelDescEnType(aclvencChannelDesc *channelDesc, acldvppStreamFormat enType);
aclError aclvencSetChannelDescPicFormat(aclvencChannelDesc *channelDesc, acldvppPixelFormat picFormat);
aclError aclvencSetChannelDescPicWidth(aclvencChannelDesc *channelDesc, uint32_t picWidth);
aclError aclvencSetChannelDescPicHeight(aclvencChannelDesc *channelDesc, uint32_t picHeight);
aclError aclvencSetChannelDescKeyFrameInterval(aclvencChannelDesc *channelDesc, uint32_t keyFrameInterval);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelDesc</td>
<td>Input</td>
<td>Description of the video encoding channel</td>
</tr>
<tr>
<td>threadId</td>
<td>Input</td>
<td>Callback thread ID</td>
</tr>
<tr>
<td>callback</td>
<td>Input</td>
<td>Encoding callback function</td>
</tr>
<tr>
<td>enType</td>
<td>Input</td>
<td>Video encoding protocol: H.265 main profile (0), H.264 baseline profile (1),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>H.264 main profile (2), or H.264 high profile (3)</td>
</tr>
<tr>
<td>picFormat</td>
<td>Input</td>
<td>Image format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● PIXEL_FORMAT_YUV_SEMIPLAN_AR_420</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● PIXEL_FORMAT_YVU_SEMIPLAN_AR_420</td>
</tr>
<tr>
<td>picWidth</td>
<td>Input</td>
<td>Image width</td>
</tr>
<tr>
<td>picHeight</td>
<td>Input</td>
<td>Image height</td>
</tr>
<tr>
<td>keyFrameInterval</td>
<td>Input</td>
<td>Key frame interval. The value cannot be 0.</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.
7.15.34.3 aclvencGetChannelDesc series

Description
Obtains the description of a video encoding channel.

Prototype

```c
uint32_t aclvencGetChannelDescChannelId(const aclvencChannelDesc *channelDesc);
uint64_t aclvencGetChannelDescThreadId(const aclvencChannelDesc *channelDesc);
aclvencCallback aclvencGetChannelDescCallback(const aclvencChannelDesc *channelDesc);
acldvppStreamFormat aclvencGetChannelDescEnType(const aclvencChannelDesc *channelDesc);
acldvppPixelFormat aclvencGetChannelDescPicFormat(const aclvencChannelDesc *channelDesc);
uint32_t aclvencGetChannelDescPicWidth(const aclvencChannelDesc *channelDesc);
uint32_t aclvencGetChannelDescPicHeight(const aclvencChannelDesc *channelDesc);
uint32_t aclvencGetChannelDescKeyFrameInterval(const aclvencChannelDesc *channelDesc)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelDesc</td>
<td>Input</td>
<td>Description of the video encoding channel</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelId</td>
<td>Channel ID (default = 0)</td>
</tr>
<tr>
<td>threadId</td>
<td>Callback thread ID</td>
</tr>
<tr>
<td>callback</td>
<td>Encoding callback function</td>
</tr>
<tr>
<td>enType</td>
<td>Video encoding protocol: H.265 main profile (0), H.264 baseline profile (1), H.264 main profile (2), or H.264 high profile (3)</td>
</tr>
<tr>
<td>picFormat</td>
<td>Input format</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>picWidth</td>
<td>Image width</td>
</tr>
<tr>
<td>picHeight</td>
<td>Image height</td>
</tr>
<tr>
<td>keyFrameInterval</td>
<td>Key frame interval. The value cannot be 0.</td>
</tr>
</tbody>
</table>

### 7.15.34 aclvencDestroyChannelDesc

**Description**

Destroys the data of the `aclvencChannelDesc` type created by calling `aclvencCreateChannelDesc`.

**Prototype**

```
aclexError aclvencDestroyChannelDesc(aclvencChannelDesc *channelDesc)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>channelDesc</td>
<td>Input</td>
<td><code>channelDesc</code> to be destroyed</td>
</tr>
</tbody>
</table>

**Returns**

0 indicates success. Other values indicate failure.

### 7.15.35 aclvencFrameConfig

Interfaces of this data type are all synchronous interfaces.

### 7.15.35.1 aclvencCreateFrameConfig

**Description**

Creates data of the `aclvencFrameConfig` type and configures the single-frame encoding configuration.

**Prototype**

```
aclvencFrameConfig *aclvencCreateFrameConfig()
```

**Parameters**

None
Returns

- Success: data of the `aclvencFrameConfig` type
- Failure: null

### 7.15.35.2 aclvencSetFrameConfig series

**Description**

Sets the single-frame encoding parameters.

**Prototype**

```
#include "aclvenc.h"

aclError aclvencSetFrameConfigForceIFrame(aclvencFrameConfig *config, uint8_t forceIFrame);

aclError aclvencSetFrameConfigEos(aclvencFrameConfig *config, uint8_t eos)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>config</td>
<td>Input</td>
<td>Single-frame encoding configuration</td>
</tr>
<tr>
<td>forceIFrame</td>
<td>Input</td>
<td>I-frame restart</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0: invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1: I-frame restarted</td>
</tr>
<tr>
<td>eos</td>
<td>Input</td>
<td>EOS flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0: invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1: EOS flag</td>
</tr>
</tbody>
</table>

**Returns**

0 indicates success. Other values indicate failure.

### 7.15.35.3 aclvencGetFrameConfig series

**Description**

Obtains the single-frame encoding parameters.

**Prototype**

```
#include "aclvenc.h"

uint8_t aclvencGetFrameConfigForceIFrame(const aclvencFrameConfig *config);

uint8_t aclvencGetFrameConfigEos(const aclvencFrameConfig *config)
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>config</td>
<td>Input</td>
<td>Single-frame encoding</td>
</tr>
<tr>
<td></td>
<td></td>
<td>configuration</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>forceIFrame</td>
<td>I-frame restart</td>
</tr>
<tr>
<td></td>
<td>● 0: invalid</td>
</tr>
<tr>
<td></td>
<td>● 1: I-frame restarted</td>
</tr>
<tr>
<td>eos</td>
<td>EOS flag</td>
</tr>
<tr>
<td></td>
<td>● 0: invalid</td>
</tr>
<tr>
<td></td>
<td>● 1: EOS flag</td>
</tr>
</tbody>
</table>

7.15.35.4 aclvencDestroyFrameConfig

Description

Destroys data of the aclvencFrameConfig type created by calling aclvencCreateFrameConfig.

Prototype

aclError aclvencDestroyFrameConfig(aclvencFrameConfig *config)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>config</td>
<td>Input</td>
<td>config to be destroyed</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.

7.15.36 aclfvRepoRange

Ascend 310 AI Processor does not support APIs described in this section.
7.15.36.1 aclfvCreateRepoRange

Description

Creates data of the `aclfvRepoRange` type for the feature repository range. It is a synchronous interface.

Prototype

```c
aclfvRepoRange *aclfvCreateRepoRange(uint32_t id0Min, uint32_t id0Max,
uint32_t id1Min, uint32_t id1Max)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id0Min</td>
<td>Input</td>
<td>Start value of id0. Must be less than or equal to id0Max within [0, 1023].</td>
</tr>
<tr>
<td>id0Max</td>
<td>Input</td>
<td>Maximum value of id0. Must be within [0, 1023].</td>
</tr>
<tr>
<td>id1Min</td>
<td>Input</td>
<td>Start value of id1. Must be less than or equal to id1Max within [0, 1023].</td>
</tr>
<tr>
<td>id1Max</td>
<td>Input</td>
<td>Maximum value of id1. Must be within [0, 1023].</td>
</tr>
</tbody>
</table>

Returns

- **Success**: data of the `aclfvRepoRange` type
- **Failure**: `null`

7.15.36.2 aclfvDestroyRepoRange

Description

Destroys the data of the `aclfvRepoRange` type created by calling `aclfvCreateRepoRange`. It is a synchronous interface.

Prototype

```c
aclError aclfvDestroyRepoRange(aclfvRepoRange *repoRange)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>repoRange</td>
<td>Input</td>
<td><code>repoRange</code> to be destroyed</td>
</tr>
</tbody>
</table>
Returns

0 indicates success. Other values indicate failure.

7.15.37 aclfvFeatureInfo

Ascend 310 AI Processor does not support APIs described in this section.

7.15.37.1 aclfvCreateFeatureInfo

Description

Creates data of the aclfvFeatureInfo type for feature description. It is a synchronous interface.

Prototype

aclfvFeatureInfo *aclfvCreateFeatureInfo(uint32_t id0, uint32_t id1, uint32_t offset, uint32_t featureLen, uint32_t featureCount, uint8_t *featureData, uint32_t featureDataLen)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id0</td>
<td>Input</td>
<td>Repository ID 0 within [0, 1023]. Defaults to 0 in the N:M scenario. id0 and id1 together identify a repository.</td>
</tr>
<tr>
<td>id1</td>
<td>Input</td>
<td>Repository ID 1 within [0, 1023]. Defaults to 0 in the N:M scenario. id0 and id1 together identify a repository.</td>
</tr>
<tr>
<td>offset</td>
<td>Input</td>
<td>Offset of the first added feature to the repository. Defaults to 0 in the N:M scenario.</td>
</tr>
<tr>
<td>featureLen</td>
<td>Input</td>
<td>Feature length, fixed at 36 bytes. The system verifies the length internally.</td>
</tr>
<tr>
<td>featureCount</td>
<td>Input</td>
<td>Number of features, up to 1 million in the 1:N scenario or up to 10 million in the N:M scenario</td>
</tr>
<tr>
<td>featureData</td>
<td>Input</td>
<td>Pointer to the feature value, stored consecutively based on the feature length</td>
</tr>
<tr>
<td>Parameter</td>
<td>Input/Output</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>featureDataLen</td>
<td>Input</td>
<td>Length of the memory allocated for the featureData pointer, used for verification</td>
</tr>
</tbody>
</table>

### Returns

- **Success**: data of the aclfvFeatureInfo type
- **Failure**: null

#### 7.15.37.2 aclfvDestroyFeatureInfo

**Description**

Destroys the data of the aclfvFeatureInfo type created by calling aclfvCreateFeatureInfo. It is a synchronous interface.

**Prototype**

```c
aclError aclfvDestroyFeatureInfo(aclfvFeatureInfo *featureInfo)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>featureInfo</td>
<td>Input</td>
<td>featureInfo to be destroyed</td>
</tr>
</tbody>
</table>

**Returns**

0 indicates success. Other values indicate failure.

#### 7.15.38 aclfvQueryTable

Ascend 310 AI Processor does not support APIs described in this section.

#### 7.15.38.1 aclfvCreateQueryTable

**Description**

Creates data of the aclfvQueryTable type for the search input table. It is a synchronous interface.

**Prototype**

```c
aclfvQueryTable *aclfvCreateQueryTable(uint32_t queryCnt, uint32_t tableLen, uint8_t *tableData, uint32_t tableDataLen)
```
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queryCnt</td>
<td>Input</td>
<td>Number of search requests, fixed at 1 in the 1:N scenario or up to 1024 in the N:M scenario</td>
</tr>
<tr>
<td>tableLen</td>
<td>Input</td>
<td>Table length, fixed at 32 KB. The system verifies the length internally.</td>
</tr>
<tr>
<td>tableData</td>
<td>Input</td>
<td>Feature value table, stored consecutively based on the feature length</td>
</tr>
<tr>
<td>tableDataLen</td>
<td>Input</td>
<td>Length of the memory allocated for the tableData pointer, used for verification</td>
</tr>
</tbody>
</table>

### Returns

- Success: data of the aclfvQueryTable type
- Failure: null

### 7.15.38.2 aclfvDestroyQueryTable

**Description**

Destroys the data of the aclfvQueryTable type created by calling aclfvCreateQueryTable. It is a synchronous interface.

**Prototype**

```c
aclError aclfvDestroyQueryTable(aclfvQueryTable *queryTable)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queryTable</td>
<td>Input</td>
<td>queryTable to be destroyed</td>
</tr>
</tbody>
</table>

**Returns**

0 indicates success. Other values indicate failure.

### 7.15.39 aclfvSearchInput

Ascend 310 AI Processor does not support APIs described in this section.
7.15.39.1 aclfvCreateSearchInput

**Description**

Creates data of the `aclfvSearchInput` type for the input of a search task. It is a synchronous interface.

**Prototype**

```c
aclfvSearchInput *aclfvCreateSearchInput(aclfvQueryTable *queryTable, aclfvRepoRange *repoRange, uint32_t topk)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queryTable</td>
<td>Input</td>
<td>Input search table</td>
</tr>
<tr>
<td>repoRange</td>
<td>Input</td>
<td>Repository range</td>
</tr>
<tr>
<td>topk</td>
<td>Input</td>
<td>Number of results to be returned per search request.</td>
</tr>
</tbody>
</table>

**Returns**

- Success: data of the `aclfvSearchInput` type
- Failure: `null`

7.15.39.2 aclfvDestroySearchInput

**Description**

Destroys the data of the `aclfvSearchInput` type created by calling `aclfvCreateSearchInput`. It is a synchronous interface.

**Prototype**

```c
aclError aclfvDestroySearchInput(aclfvSearchInput *searchInput)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>searchInput</td>
<td>Input</td>
<td><code>searchInput</code> to be destroyed</td>
</tr>
</tbody>
</table>

**Returns**

0 indicates success. Other values indicate failure.
7.15.40 aclfvSearchResult

Ascend 310 AI Processor does not support APIs described in this section.

7.15.40.1 aclfvCreateSearchResult

Description

Creates data of the aclfvSearchResult type for the search result. It is a synchronous interface.

Prototype

```
aclfvSearchResult *aclfvCreateSearchResult(uint32_t queryCnt, uint32_t *resultNum, uint32_t resultNumDataLen, uint32_t *id0, uint32_t *id1, uint32_t *resultOffset, float *resultDistance, uint32_t dataLen)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queryCnt</td>
<td>Input</td>
<td>Number of search requests, fixed at 1 in the 1:N scenario</td>
</tr>
<tr>
<td>resultNum</td>
<td>Input</td>
<td>Number of search results</td>
</tr>
<tr>
<td>resultNumDataLen</td>
<td>Input</td>
<td>Total memory size of resultNum</td>
</tr>
<tr>
<td>id0</td>
<td>Input</td>
<td>Repository ID 0. The number of elements equals topK * queryCnt.</td>
</tr>
<tr>
<td>id1</td>
<td>Input</td>
<td>Repository ID 1. The number of elements equals topK * queryCnt.</td>
</tr>
<tr>
<td>resultOffset</td>
<td>Input</td>
<td>Offset of the repository corresponding to the search results of each search request. The number of elements equals topK * queryCnt.</td>
</tr>
<tr>
<td>resultDistance</td>
<td>Input</td>
<td>Distance between each search result and the search request. The number of elements equals topK * queryCnt.</td>
</tr>
<tr>
<td>dataLen</td>
<td>Input</td>
<td>Requested memory size: topK * queryCnt * 4 bytes</td>
</tr>
</tbody>
</table>
Returns

- Success: data of the aclfvSearchResult type
- Failure: null

7.15.40.2 aclfvDestroySearchResult

Description

Destroys the data of the aclfvSearchResult type created by calling aclfvCreateSearchResult. It is a synchronous interface.

Prototype

```
aclError aclfvDestroySearchResult(aclfvSearchResult *searchResult)
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Input/Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>searchResult</td>
<td>Input</td>
<td>searchResult to be destroyed</td>
</tr>
</tbody>
</table>

Returns

0 indicates success. Other values indicate failure.
8 ACL Sample Description

8.1 Samples
8.2 Matrix-matrix multiplication
8.3 Image Classification based on Caffe ResNet-50 (Including Image Decoding and Resizing)
8.4 Image Classification based on Caffe ResNet-50 (Including Image Decoding, Cropping, Resizing, and Encoding)
8.5 Image Classification Based on Caffe ResNet-50 (Including Video Decoding)
8.6 Image Classification Based on Caffe ResNet-50 (Synchronous Inference, Excluding Data Pre-Processing)
8.7 Image Classification Based on Caffe ResNet-50 (Asynchronous Inference, Excluding Data Pre-Processing)
8.8 CompileDemo
8.9 InferClassification

8.1 Samples

The section about ACL features provides the sample code description. After the development environment is deployed, some samples are generated in the installation directory of the development kit. For example, if the installation directory of the development kit is `/home/HwHiAiuser/ascend-toolkit/`, the sample code is stored in the `/home/HwHiAiuser/ascend-toolkit/latest/x.x.x-linux_gccx.x.x/acllib/sample` directory. See the following table.

<table>
<thead>
<tr>
<th>Sample Name</th>
<th>Directory</th>
<th>Basic Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>acl_execute_gemm</td>
<td>acl_execute_op</td>
<td>Matrix-matrix multiplication</td>
</tr>
</tbody>
</table>
### Sample Name | Directory | Basic Function
--- | --- | ---
`acl_dvpp_resnet50` | `acl_execute_model` | Image classification based on Caffe ResNet-50 (including image decoding and resizing)

`acl_vpc_jpege_resnet50` | `acl_execute_model` | Image classification based on Caffe ResNet-50 (including image decoding, cropping, resizing, and encoding)

`acl_vdec_resnet50` | `acl_execute_model` | Image Classification Based on Caffe ResNet-50 (Including Video Decoding)

`acl_resnet50` | `acl_execute_model` | Image Classification Based on Caffe ResNet-50 (Synchronous Inference, Excluding Data Pre-Processing)

`acl_resnet50_async` | `acl_execute_model` | Image Classification Based on Caffe ResNet-50 (Asynchronous Inference, Excluding Data Pre-Processing)

**NOTE**

The sample directory contains a DVPP sample, which is not supported currently.

To improve the code development efficiency, some code samples and tools are open-sourced for the software components and inference services of Ascend AI processors. You can log in to [https://www.huaweicloud.com/ascend/resources/CodeSamples](https://www.huaweicloud.com/ascend/resources/CodeSamples) to obtain the code samples and tools. This document describes the code development process and principles based on the code samples listed in Table 8-1.
8.2 Matrix-matrix multiplication

8.2.1 Sample Overview

Function Description

This sample implements the matrix-matrix multiplication operation: \( C = \alpha AB + \beta C \). In this operation, \( A, B, \) and \( C \) are \( 16 \times 16 \) matrices, meaning that \( m=16, n=16, \) and \( k=16 \). The result is a \( 16 \times 16 \) matrix.

![Sample diagram](image)

Basic Principles

The following table lists the key functions involved in this sample. For details about APIs, see See Also > ACL API Reference.
Table 8-2 Key functions

| Initialization | • Call aclInit to initialize the ACL configuration.  
|                | • Call aclFinalize to deinitialize the ACL configuration. |
| Device Management | • Call aclrtSetDevice to specify the device for computation.  
|                  | • Call aclrtGetRunMode to obtain the running mode of the Ascend AI Software Stack. The internal processing process varies according to the running mode.  
|                  | • Call aclrtResetDevice to reset the current device and reclaim the resources on the device. |
| Streams Management | • Call aclrtCreateStream to create a stream.  
|               | • Call aclrtDestroyStream to destroy a stream.  
|                | • Call aclrtSynchronizeStream to block the host processing until all tasks in the specified stream are completed. |
| Memory Management | • Call aclrtMallocHost to allocate memory on the host.  
|                | • Call aclrtFreeHost to free memory on the host.  
|                | • Call aclrtMalloc to allocate memory on the device.  
|                | • Call aclrtFree to free the memory on the device. |
| Data Movement | Call aclrtMemcpy to:  
|               | • Copy data from the host to the device as the input data for matrix multiplication.  
|               | • Copy the matrix multiplication result from the device to the host. |
| Single-Operator Calling | • Call aclblasGemmEx to implement matrix-matrix multiplication. Users can specify the data types of the elements in the matrices. A matrix multiplication operator Gemm has been encapsulated in the aclblasGemmEx API.  
|               | • Use the Ascend Tensor Compiler (ATC) to build the operator description information (including the input and output tensor description and operator attributes) of the built-in matrix multiplication operator Gemm into an offline model adapted to the Ascend AI Processor (an *.om file) to verify the operator running result. |

Directory Structure

The code structure of the acl_execute_gemm sample is as follows:

```
    ├── inc
    │   ├── common.h          //Header file that declares common functions (such as the file read function)  
    │   └── gemm_runner.h     //Header file that declares the functions related to matrix multiplication  
    └── run
        └── test_data
            └── config
```
8.2.2 Building and Running Application on Host

This document uses HwHiAiUser as an example.

**Step 1** Convert your model.

1. Log in to the development environment as the HwHiAiUser user (running user).
2. Obtain the tool and set environment variables. For details, see 5.3 Model Conversion Using ATC.
3. Compile the operator description information (*.json file) of the matrix multiplication operator into an offline model adapted to the Ascend AI Processor (*.om file) for running the matrix multiplication operator.

   Go to the acl_execute_gemm directory and run the following command:
   ```
   atc --singleop=run/out/test_data/config/gemm.json --soc_version=Ascend310 --output=run/out/op_models
   ````
   
   **--output**: specifies the directory for storing the generated OM file, that is, the run/out/op_models directory.

   For details about the parameters, see Restrictions and Parameters in ATC Tool Instructions.

**Step 2** Compile the code.

1. Log in to the development environment as the HwHiAiUser user (running user).
2. Go to the acl_execute_gemm/run/out/test_data/data directory and run the generate_data.py script. The data files matrix_a.bin of matrix A, matrix_b.bin of matrix B, and matrix_c.bin of matrix C are generated in the run/out/test_data/data directory.

   ```
   python3.7.5 generate_data.py
   ```

   **NOTE**

   Data in the output.bin file generated in the run/out/test_data/data directory is the matrix multiplication result calculated using the formula in the generate_data.py script and is not used as the input data of this sample.

3. Set environment variables. The build script src/CMakeLists.txt builds code based on the header file path and library file path set by environment variables.

   The build script CMakeLists.txt is predefined with a default header file path (/usr/local/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/include/) and a default library file path (/usr/local/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/lib64/stub/).

   If the S{DDK_PATH} and S{NPU_HOST_LIB} environment variables are not specified, the paths predefined in the build script CMakeLists.txt are used. If
the `$DDK_PATH` and `$NPU_HOST_LIB` environment variables have been set, the build script `CMakeLists.txt` looks up the header files and library files that the build depends on in the paths specified by the environment variables.

The following is an example of setting environment variables. Replace `/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x` with the actual ACLlib path.

**Perform one of the following steps based on the actual situation:**

If the OS architecture of the development environment is the same as that of the operating environment, run the following commands to import environment variables:

```bash
export DDK_PATH=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x
export NPU_HOST_LIB=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/lib64/stub
```

When the operating environment is deployed on Atlas 500 AI edge stations, if the development environment uses the x86 architecture and the operating environment uses the ARM architecture, the ACLlib library of the ARM architecture development kit needs to be called during application compilation because the development kits of both the x86 and ARM architectures are deployed in the development environment. Therefore, you need to import the environment variables to the path of the ACLlib library in the ARM architecture.

```bash
export DDK_PATH=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/arm64-linux_gcc7.3.0
export NPU_HOST_LIB=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/arm64-linux_gcc7.3.0/acllib/lib64/stub
```

**NOTE**

The `.so` library files in the `/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/lib64/stub` directory are required to build the code logic based on the ACL APIs, without depending on any `.so` library files of other components (such as Driver).

After successful build, when you run an application on the host, the application can be linked to the `.so` library files in the `/home/HwHiAiUser/Ascend/nmrt/latest/xxx-linux_gccx.x.x/acllib/lib64` directory on the host by configuring corresponding environment variables. The application is automatically linked to the dependent `.so` library files of other components during running.

4. Go to the `acl_execute_gemm` directory and create a directory for storing build outputs. For example, the directory created in this sample is `build/intermediates/host`.

   ```bash
   mkdir -p build/intermediates/host
   ```

5. Go to the `build/intermediates/host` directory and run the `cmake` command.

   ```bash
   .././../src
   ```

Replace it with the actual path.

```bash
cd build/intermediates/host
```

If the OS architecture of the development environment is the same as that of the operating environment, run the following command to perform compilation:

```bash
make ../../src -DCMAKE_CXX_COMPILER=g++ -DCMAKE_SKIP_RPATH=TRUE
```

If the operating environment is deployed on an Atlas 500 AI edge station, the development environment uses the x86 architecture, and the operating environment uses the ARM architecture, run the following command to perform cross compilation:

```bash
make ../../src -DCMAKE_CXX_COMPILER=aarch64-linux-gnu-g++ -DCMAKE_SKIP_RPATH=TRUE
```

6. Run the `make` command. The executable file `execute_gemm_op` is generated in the `acl_execute_gemm/run/out` directory.

   ```bash
   make
   ```

**Step 3** Run the application.
In the development and commissioning scenario, a compiled application is running in the development environment. Therefore, you need to import the following environment variables to the development environment:

```
export LD_LIBRARY_PATH=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/lib64
```

Then perform Step 3.5 to run the application.

In the operating scenario, you need to perform Step 3.1 to Step 3.5 to copy the application compiled in the development environment to the operating environment, and then run the application.

1. Upload all files in the `acl_execute_gemm/run/out` directory in the development environment to any directory in the operating environment (host) as the HwHiAiUser user (execution user), for example, `/home/HwHiAiUser/acl_execute_gemm`.

2. Log in to the operating environment (host) as the HwHiAiUser user (running user).

3. Set the environment variable.
   The following is an example only. Replace the path with the actual one.
   ```
   export LD_LIBRARY_PATH=/home/HwHiAiUser/Ascend/nnrt/latest/xxx-linux_gccx.x.x/acllib/lib64:/usr/local/Ascend/add-ons
   ```

4. Go to the directory where the executable file `execute_gemm_op` is located (for example, `/home/HwHiAiUser/acl_execute_gemm/`), and grant the execute permission on the `execute_gemm_op` file in the directory.
   ```
   chmod +x execute_gemm_op
   ```

5. Go to the directory where the executable file `execute_gemm_op` is located (for example, `/home/HwHiAiUser/acl_execute_gemm/`), and run the executable file.
   ```
   ./execute_gemm_op
   ```

After the command is executed successfully, the data of matrix A and matrix B, and the matrix multiplication result are displayed in the terminal window. In addition, a `matrix_c.bin` file that stores the matrix multiplication result is generated in the `result_files` directory.

---

8.3 Image Classification based on Caffe ResNet-50 (Including Image Decoding and Resizing)

8.3.1 Sample Overview

Function Description

This sample shows how to classify images based on the Caffe ResNet-50 network (single input and single batch).

Convert the model file of the Caffe ResNet-50 network into an offline model adapted to the Ascend AI Processor (an *.om file). In the sample, load the *.om file, decode, resize and infer two *.jpg images, obtain the inference results, process the inference results, and output the category ID with the maximum confidence.
During model conversion, you need to set the CSC parameters to convert YUV420SP images into RGB images to meet the input requirements of the model.

**Figure 8-2** Sample diagram

![Sample diagram](image)

**Basic Principles**

The following table lists the key functions involved in this sample. For details about APIs, see See Also > ACL API Reference.

| Initialization       | - Call `aclInit` to initialize the ACL configuration.  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- Call <code>aclFinalize</code> to deinitialize the ACL configuration.</td>
</tr>
</tbody>
</table>

| Device Management | - Call `aclrtSetDevice` to specify the device for computation.  
|                  | - Call `aclrtGetRunMode` to obtain the running mode of the Ascend AI Software Stack. The internal processing process varies according to the running mode.  
|                  | - Call `aclrtResetDevice` to reset the current device and reclaim the resources on the device. |
Context Management
- Call `aclrtCreateContext` to create a context.
- Call `aclrtDestroyContext` to destroy the context.

Streams Management
- Call `aclrtCreateStream` to create a stream.
- Call `aclrtDestroyStream` to destroy a stream.
- Call `aclrtSynchronizeStream` to block the host processing until all tasks in the specified stream are completed.

Memory Management
- Call `aclrtMallocHost` to allocate memory on the host.
- Call `aclrtFreeHost` to free memory on the host.
- Call `aclrtMalloc` to allocate memory on the device.
- Call `aclrtFree` to free the memory on the device.

At data preprocessing, if you need to allocate the memory on the device to store the input or output data, call `acldvppMalloc` to allocate memory and call `acldvppFree` to free the memory.

Data Movement
Call `aclrtMemcpy` to:
- Copy data from the host to the device as the input data for decoding.
- Copy the inference result from the device to the host.

Data Preprocessing
- Call `acldvppJpegDecodeAsync` to decode *.jpg images into YUV420SP images.
- Call `acldvppVpcResizeAsync` to resize a YUV420SP image to a 224 x 224 image.

Model Inference
- Call `aclmdlLoadFromFileWithMem` to load the model from the *.om file.
- Call `aclmdlExecute` to perform model inference. Before inference, use the CSC parameters in the *.om file to convert a YUV420SP image into an RGB image.
- Call `aclmdlUnload` to unload the model.

Data Post-Processing (Single Operator Calling)
Process the model inference result. Call `Cast` to convert the data type of the inference result from float32 to float16, and then call `ArgMaxD` to search for the category ID with the maximum confidence in the inference result.
Call `aclopExecute` to load and execute operators.

Directory Structure
The code structure of the acl_dvpp_resnet50 sample is as follows:
```
├acl_dvpp_resnet50
│   └── caffe_model
│       ├── aipp.cfg //Configuration file with CSC parameters, used for model conversion
│       └── resnet50.prototxt //Model file of the ResNet50 network
└── data
    └── dog1_1024_683.jpg //Test data
```
8.3.2 Building and Running Application on Host

This document uses *HwHiAiUser* as an example.

**Step 1** Convert your model.

1. Log in to the development environment as the *HwHiAiUser* user (running user).
2. Obtain the tool and set environment variables. For details, see 5.3 Model Conversion Using ATC.
3. Prepare data.

Obtain the weight file (*.caffemodel) of the ResNet-50 network from the [https://gitee.com/HuaweiAscend/models/tree/master/computer_vision/classification/resnet50](https://gitee.com/HuaweiAscend/models/tree/master/computer_vision/classification/resnet50) and upload the file to the *acl_dvpp_resnet50/caffe_model* directory in the development environment as the *HwHiAiUser* user (execution user).

4. Convert the ResNet-50 network to an offline model adapted to the Ascend AI Processor (an *.om file). During model conversion, you need to set CSC parameters to convert YUV420SP images to RGB images.

Go to the *acl_dvpp_resnet50* directory and run the following command:

```
    atc --model=caffe_model/resnet50.prototxt --weight=caffe_model/resnet50.caffemodel --framework=0 --output=model/resnet50_aipp --soc_version=Ascend310 --insert_op_conf=caffe_model/aipp.cfg
```

**--output:** The generated *resnet50_aipp.om* file is stored in the *acl_dvpp_resnet50/model* directory. The default settings are recommended in the command. Otherwise, you need to change the value of *omModelPath* in *sample_process.cpp* before building the code.

```
    const char* omModelPath = "./model/resnet50_aipp.om";
```

For details about the parameters, see Restrictions and Parameters in ATC Tool Instructions.
5. Build the operator description information (.json files) of the Cast and ArgMaxD operators into an offline model adapted to the Ascend AI Processor (an *.om file) separately to run the operator.

Go to the acl_dvpp_resnet50 directory and run the following command:
```
atc --singleop=out/op_models/op_list.json --soc_version=Ascend310 --output=out/op_models
```

---output specifies the directory for storing the generated OM file, that is, the out/op_models directory.

For details about the parameters, see Restrictions and Parameters in ATC Tool Instructions.

**Step 2 Compile the code.**

1. Log in to the development environment as the HwHiAiUser user (running user).
2. Set environment variables. The build script src/CMakeLists.txt builds code based on the header file path and library file path set by environment variables.

The build script CMakeLists.txt is predefined with a default header file path (/usr/local/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/include/) and a default library file path (/usr/local/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/lib64/stub/).

If the ${DDK_PATH} and ${NPU_HOST_LIB} environment variables are not specified, the paths predefined in the build script CMakeLists.txt are used. If the ${DDK_PATH} and ${NPU_HOST_LIB} environment variables have been set, the build script CMakeLists.txt looks up the header files and library files that the build depends on in the paths specified by the environment variables.

The following is an example of setting environment variables. Replace /home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x with the actual ACLlib path.

Perform one of the following steps based on the actual situation:

If the OS architecture of the development environment is the same as that of the operating environment, run the following commands to import environment variables:
```
export DDK_PATH=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x
export NPU_HOST_LIB=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/lib64/stub
```

When the operating environment is deployed on Atlas 500 AI edge stations, if the development environment uses the x86 architecture and the operating environment uses the ARM architecture, the ACLlib library of the ARM architecture development kit needs to be called during application compilation because the development kits of both the x86 and ARM architectures are deployed in the development environment. Therefore, you need to import the environment variables to the path of the ACLlib library in the ARM architecture.
```
export DDK_PATH=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/arm64-linux_gcc7.3.0
export NPU_HOST_LIB=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/arm64-linux_gcc7.3.0/acllib/lib64/stub
```
NOTE

The .so library files in the /home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acclib/lib64/stub directory are required to build the code logic based on the ACL APIs, without depending on any .so library files of other components (such as Driver).

After successful build, when you run an application on the host, the application can be linked to the .so library files in the /home/HwHiAiUser/Ascend/nnrt/latest/xxx-linux_gccx.x.x/acclib/lib64 directory on the host by configuring corresponding environmental variables. The application is automatically linked to the dependent .so library files of other components during running.

3. Go to the acl_dvpp_resnet50 directory and create a directory for storing build outputs. For example, the directory created in this sample is build/intermediates/host.

```bash
mkdir -p build/intermediates/host
```

4. Go to the build/intermediates/host directory and run the cmake command.

```bash
.././../src indicates the directory where the CMakeLists.txt file is located. Replace it with the actual path.
```

```bash
cd build/intermediates/host
```

If the OS architecture of the development environment is the same as that of the operating environment, run the following command to perform compilation:

```bash
cmake ../../src -DCMAKE_CXX_COMPILER=g++ -DCMAKE_SKIP_RPATH=TRUE
```

If the operating environment is deployed on an Atlas 500 AI edge station, the development environment uses the x86 architecture, and the operating environment uses the ARM architecture, run the following command to perform cross compilation:

```bash
cmake ../../src -DCMAKE_CXX_COMPILER=aarch64-linux-gnu-g++ -DCMAKE_SKIP_RPATH=TRUE
```

5. Run the make command to generate an executable file main in the acl_dvpp_resnet50/out directory.

```bash
make
```

### Step 3 Run the application.

NOTE

- In the development and commissioning scenario, a compiled application is running in the development environment. Therefore, you need to import the following environment variables to the development environment:

  ```bash
  export LD_LIBRARY_PATH=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acclib/lib64
  
  Then perform Step 3.5 to run the application.
  ```

- In the operating scenario, you need to perform Step 3.1 to Step 3.5 to copy the application compiled in the development environment to the operating environment, and then run the application.

1. Upload the src, out, data, and model directories in the acl_dvpp_resnet50 directory in the development environment to the same directory in the operating environment (host) as the HwHiAiUser user (execution user), for sample, /home/HwHiAiUser/acl_dvpp_resnet50.

2. Log in to the operating environment (host) as the HwHiAiUser user (running user).

3. Set the environment variable.

   ```bash
   export LD_LIBRARY_PATH=/home/HwHiAiUser/Ascend/nnrt/latest/xxx-linux_gccx.x.x/acclib/lib64:/usr/local/Ascend/add-ons
   ```

4. Go to the directory where the executable file main is located (for example, /home/HwHiAiUser/acl_dvpp_resnet50/out) and grant the execute permission on the main file in the directory.
5. Go to the directory where the executable file `main` is located (for sample, `/home/HwHiAiUser/acl_dvpp_resnet50/out`) and run the executable file.

```
./main
```

After the command is executed successfully, the category ID with the maximum confidence is displayed.

```
INFO acl init success
INFO open device 0 success
INFO create context success
INFO create stream success
INFO dvpp init resource success
INFO load model ./test_data/model/resnet50_aipp.om success
INFO create model description success
INFO create model output success
INFO start to process picture:./test_data/dog1_1024_683.jpg
INFO Process dvpp success
INFO model execute success
INFO execute singleOp Cast success
INFO execute ArgMax0 success
INFO singleOp process success
INFO --> index of classification result is 181
INFO start to process picture:./test_data/dog2_1024_683.jpg
INFO Process dvpp success
INFO model execute success
INFO execute singleOp Cast success
INFO execute ArgMax0 success
INFO singleOp process success
INFO --> index of classification result is 207
INFO unload model success, modelid is 1
INFO end to destroy stream
INFO end to destroy context
INFO end to reset device is 0
```
According to the arguments of the app, the following functions can be implemented:

- Encode a YUV420SP image into a *.jpg image.
- Decodes two *.jpg images into YUV420SP NV12 images, resizes the images, performs model inference to obtain the inference results of the two images, processes the inference results, and outputs the category IDs with the corresponding maximum confidences and the sum of the top 5 confidences.
- Decodes two *.jpg images into YUV420SP NV12 images, crops the images, performs model inference to obtain the inference results of the two images, processes the inference results, and outputs the category IDs with the corresponding maximum confidences and the sum of the top 5 confidences.
- Decodes two *.jpg images into YUV420SP NV12 images, crops the images and overwrites the target images with the cropped areas, performs model inference to obtain the inference results of the two images, processes the inference results, and outputs the category IDs with the corresponding maximum confidences and the sum of the top 5 confidences.

In this sample, an *om file (offline model adapted to the Ascend AI Processor) is used for inference. During model conversion, you need to set the CSC parameters to convert YUV420SP images into RGB images to meet the model input requirements.

Principles

The following table lists the key functions involved in this sample. For details about APIs, see See Also > ACL API Reference.

<table>
<thead>
<tr>
<th>Initialization</th>
<th>Call aclInit to initialize the ACL configuration.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Call aclFinalize to deinitialize the ACL configuration.</td>
</tr>
<tr>
<td>Device Management</td>
<td>Call aclrtSetDevice to specify the device for computation.</td>
</tr>
<tr>
<td></td>
<td>Call aclrtGetRunMode to obtain the running mode of the Ascend AI Software Stack. The internal processing process varies according to the running mode.</td>
</tr>
<tr>
<td></td>
<td>Call aclrtResetDevice to reset the current device and reclaim the resources on the device.</td>
</tr>
<tr>
<td>Context Management</td>
<td>Call aclrtCreateContext to create a context.</td>
</tr>
<tr>
<td></td>
<td>Call aclrtDestroyContext to destroy the context.</td>
</tr>
<tr>
<td>Streams Management</td>
<td>Call aclrtCreateStream to create a stream.</td>
</tr>
<tr>
<td></td>
<td>Call aclrtDestroyStream to destroy a stream.</td>
</tr>
<tr>
<td></td>
<td>Call aclrtSynchronizeStream to block the host processing until all tasks in the specified stream are completed.</td>
</tr>
</tbody>
</table>
| Memory Management | Call aclrtMallocHost to allocate memory on the host.  
|                    | Call aclrtFreeHost to free memory on the host.  
|                    | Call aclrtMalloc to allocate memory on the device.  
|                    | Call aclrtFree to free the memory on the device.  
|                    | At data preprocessing, if you need to allocate the memory on the device to store the input or output data, call acldvppMalloc to allocate memory and call acldvppFree to free the memory. |
| Data Movement      | Call aclrtMemcp to:  
|                    | ● Copy data from the host to the device as the input data for decoding.  
|                    | ● Copy the inference result from the device to the host. |
| Data Pre-processing| ● Image Encoding  
|                    | Call acldvppJpegEncodeAsync to encode a YUV420SP image into a *.jpg image.  
|                    | ● Image Decoding  
|                    | Call acldvppJpegDecodeAsync to decode *.jpg images into YUV420SP images.  
|                    | ● Resizing  
|                    | Call acldvppVpcResizeAsync to resize a YUV420SP image to a 224 x 224 image.  
|                    | ● Cropping  
|                    | Call acldvppVpcCropAsync to crop the specified area of the input image, and then overwrite the cropped area to the output buffer as the output image.  
|                    | ● Cropping and Pasting  
|                    | Call acldvppVpcCropAndPasteAsync to crop the specified area of the input image, and then overwrite the cropped area to the specified area of the target image as the output image. |
| Model Inference    | ● Call aclmdlLoadFromFileWithMem to load the model from the *.om file.  
|                    | ● Call aclmdlExecute to perform model inference.  
|                    | Before inference, use the CSC parameters in the *.om file to convert a YUV420SP image into an RGB image.  
|                    | ● Call aclmdlUnload to unload the model. |

**Directory Structure**

The code structure of the acl_vpc_jpege_resnet50 sample is as follows:

```
├acl_vpc_jpege_resnet50
│   ├── caffe_model
│   │   ├── aipp.cfg //Configuration file with CSC parameters, used for model conversion
│   │   └── resnet50.prototxt //Model file of the ResNet50 network
│   └── data
│       ├── persian_cat_1024_1536_283.jpg //Test data
│       ├── wood_rabbit_1024_1061_330.jpg //Test data
│       └── wood_rabbit_1024_1068_nv12.yuv //Test data
```
8.4.2 Building and Running Application on Host

This document uses HwHiAiUser as an example.

**Step 1** Convert your model.

1. Log in to the development environment as the HwHiAiUser user (running user).
2. Obtain the tool and set environment variables. For details, see 5.3 Model Conversion Using ATC.
3. Prepare data.
   Obtain the weight file (*.caffemodel) of the ResNet-50 network from the [https://gitee.com/HuaweiAscend/models/tree/master/computer_vision/classification/resnet50/caffe_model](https://gitee.com/HuaweiAscend/models/tree/master/computer_vision/classification/resnet50/caffe_model) and upload the file to the acl_vpc_jpege_resnet50/caffe_model directory in the development environment as the HwHiAiUser user (execution user).
4. Convert the ResNet-50 network to an offline model adapted to the Ascend AI Processor (an *.om file). During model conversion, you need to set CSC parameters to convert YUV420SP images to RGB images.

   Go to the acl_vpc_jpege_resnet50 directory and run the following command:
   ```
   atc --model=caffe_model/resnet50.prototxt --weight=caffe_model/resnet50.caffemodel --framework=0 --output=model/resnet50_aipp --soc_version=Ascend310 --insert_op_conf=caffe_model/aipp.cfg
   ``
   
   **--output**: The generated resnet50_aipp.om file is stored in the acl_vpc_jpege_resnet50/model directory. The default settings are recommended in the command. Otherwise, you need to change the value of omModelPath in sample_process.cpp before building the code.
   ```
   const char* omModelPath = "../model/resnet50_aipp.om";
   ```
   For details about the parameters, see Restrictions and Parameters in ATC Tool Instructions.

**Step 2** Compile the code.

1. Log in to the development environment as the HwHiAiUser user (running user).
2. Set environment variables. The build script `src/CMakeLists.txt` builds code based on the header file path and library file path set by environment variables.

The build script `CMakeLists.txt` is predefined with a default header file path `(/usr/local/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/include/)` and a default library file path `(/usr/local/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/lib64/stub/)`.

If the `${DDK_PATH}` and `${NPU_HOST_LIB}` environment variables are not specified, the paths predefined in the build script `CMakeLists.txt` are used. If the `${DDK_PATH}` and `${NPU_HOST_LIB}` environment variables have been set, the build script `CMakeLists.txt` looks up the header files and library files that the build depends on in the paths specified by the environment variables.

The following is an example of setting environment variables. Replace `/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x` with the actual ACLlib path.

Perform one of the following steps based on the actual situation:

If the OS architecture of the development environment is the same as that of the operating environment, run the following commands to import environment variables:

```bash
export DDK_PATH=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x
export NPU_HOST_LIB=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/lib64/stub
```

When the operating environment is deployed on Atlas 500 AI edge stations, if the development environment uses the x86 architecture and the operating environment uses the ARM architecture, the ACLlib library of the ARM architecture development kit needs to be called during application compilation because the development kits of both the x86 and ARM architectures are deployed in the development environment. Therefore, you need to import the environment variables to the path of the ACLlib library in the ARM architecture.

```bash
export DDK_PATH=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/arm64-linux_gcc7.3.0
export NPU_HOST_LIB=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/arm64-linux_gcc7.3.0/acllib/lib64/stub
```

**NOTE**

The `.so` library files in the `/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/lib64/stub` directory are required to build the code logic based on the ACL APIs, without depending on any `.so` library files of other components (such as Driver).

After successful build, when you run an application on the host, the application can be linked to the `.so` library files in the `/home/HwHiAiUser/Ascend/nmrt/latest/xxx-linux_gccx.x.x/acllib/lib64` directory on the host by configuring corresponding environmental variables. The application is automatically linked to the dependent `.so` library files of other components during running.

3. Go to the `acl_vpc_jpege_resnet50` directory and create a directory for storing build outputs. For example, the directory created in this sample is `build/intermediates/host`.

```bash
mkdir -p build/intermediates/host
```

4. Go to the `build/intermediates/host` directory and run the `cmake` command.

`.. '/../src` indicates the directory where the `CMakeLists.txt` file is located. Replace it with the actual path.

```bash
cd build/intermediates/host
```

If the OS architecture of the development environment is the same as that of the operating environment, run the following commands to import environment variables:

```bash
export DDK_PATH=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x
export NPU_HOST_LIB=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/lib64/stub
```
environment, run the following command to perform compilation:
```bash
cmake ../../../src -DCMAKE_CXX_COMPILER=g++ -DCMAKE_SKIP_RPATH=TRUE
```
If the operating environment is deployed on an Atlas 500 AI edge station, the development environment uses the x86 architecture, and the operating environment uses the ARM architecture, run the following command to perform cross compilation:
```bash
cmake ../../../src -DCMAKE_CXX_COMPILER=aarch64-linux-gnu-g++ -DCMAKE_SKIP_RPATH=TRUE
```

5. Run the `make` command to generate an executable file `main` in the `acl_vpc_jpege_resnet50/out` directory.

```
make
```

**Step 3** Run the application.

**NOTE**

- In the development and commissioning scenario, a compiled application is running in the development environment. Therefore, you need to import the following environment variables to the development environment:
  ```bash
  export LD_LIBRARY_PATH=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/lib64
  ```
  Then perform **Step 3.5** to run the application.
- In the operating scenario, you need to perform **Step 3.1** to **Step 3.5** to copy the application compiled in the development environment to the operating environment, and then run the application.

1. Upload the `src`, `out`, `data`, and `model` directories in the `acl_vpc_jpege_resnet50` directory in the development environment to the same directory in the operating environment (host) as the `HwHiAiUser` user (execution user), for sample, `/home/HwHiAiUser/acl_vpc_jpege_resnet50`.

2. Log in to the operating environment (host) as the `HwHiAiUser` user (running user).

3. Set the environment variable.
   The following is an example only. Replace the path with the actual one.
   ```bash
   export LD_LIBRARY_PATH=/home/HwHiAiUser/Ascend/nnrt/latest/xxx-linux_gccx.x.x/acllib/lib64:/usr/local/Ascend/add-ons
   ```

4. Go to the directory where the executable file `main` is located (for example, `/home/HwHiAiUser/acl_vpc_jpege_resnet50/out`) and grant the execute permission on the `main` file in the directory.
   ```bash
   chmod +x main
   ```

5. Go to the directory where the executable file `main` is located (for sample, `/home/HwHiAiUser/acl_vpc_jpege_resnet50/out`) and run the executable file.
   a. Decode two *.jpg images into two YUV420SP NV12 images, resize them, and perform model inference to obtain the inference results of the two images.
      ```bash
      ./main 0
      ```
b. Decode two *.jpg images into two YUV420SP NV12 images, crop them, and perform model inference to obtain the inference results of the two images.

```bash
./main 1
```

**Figure 8-3 Execution result**

```bash
[INFO] acl init success
[INFO] open device 0 success
[INFO] create context success
[INFO] create stream success
[INFO] dvp init resource success
[INFO] load model ./model/resnet50_aipp.on success
[INFO] create model description success
[INFO] create model output success
[INFO] start to process picture:../data/persian_cat_1024_1536_283.jpg
[INFO] call jpeg0
[INFO] call vpcResize
[INFO] Process dvp success
[INFO] model execute success
[INFO] result : classType[283], top1[0.506486], top5[0.610396]
[INFO] start to process picture:../data/wood_rabbit_1024_1061_330.jpg
[INFO] call jpeg0
[INFO] call vpcResize
[INFO] Process dvp success
[INFO] model execute success
[INFO] result : classType[330], top1[0.542480], top5[0.100063]
[INFO] unload model success, modelId is 1
[INFO] execute sample success
[INFO] end to destroy stream
[INFO] end to destroy context
[INFO] end to reset device is 0
```

c. Decode two *.jpg images into two YUV420SP NV12 images, crop and paste them, and perform model inference to obtain the inference results of the two images.

```bash
./main 2
```

**Figure 8-4 Execution result**

```bash
[INFO] acl init success
[INFO] open device 0 success
[INFO] create context success
[INFO] create stream success
[INFO] dvp init resource success
[INFO] load model ./model/resnet50_aipp.on success
[INFO] create model description success
[INFO] create model output success
[INFO] start to process picture:../data/persian_cat_1024_1536_283.jpg
[INFO] call jpeg0
[INFO] call vpcCrop
[INFO] Process dvp success
[INFO] model execute success
[INFO] result : classType[284], top1[0.961914], top5[0.999743]
[INFO] start to process picture:../data/wood_rabbit_1024_1061_330.jpg
[INFO] call jpeg0
[INFO] call vpcCrop
[INFO] Process dvp success
[INFO] model execute success
[INFO] result : classType[330], top1[0.631836], top5[0.999888]
[INFO] unload model success, modelId is 1
[INFO] execute sample success
[INFO] end to destroy stream
[INFO] end to destroy context
[INFO] end to reset device is 0
```
Figure 8-5 Execution result

```
[INFO] acl init success
[INFO] open device 0 success
[INFO] create context success
[INFO] create stream success
[INFO] dvp init resource success
[INFO] load model ./model/resnet50_aipp.on success
[INFO] create model description success
[INFO] create model output success

[INFO] start to process picture: ./data/persian_cat_1024_1536_283.jpg
[INFO] call Jpeg0
[INFO] call vpcropAndPaste
[INFO] Process dvp success
[INFO] model execute success
[INFO] result : classType[284], top1[0.483398], top5[0.855194]

[INFO] start to process picture: ./data/wood_rabbit_1024_1661_336.jpg
[INFO] call Jpeg0
[INFO] call vpcropAndPaste
[INFO] Process dvp success
[INFO] model execute success
[INFO] result : classType[331], top1[0.670898], top5[0.953564]

[INFO] unload model success, modelId is 1
[INFO] execute sample success
[INFO] end to destroy stream
[INFO] end to destroy context
[INFO] end to reset device is 0
```

d. Encode a YUV420SP image into a *.jpg image.

Figure 8-6 Execution result

```
[INFO] acl init success
[INFO] open device 0 success
[INFO] create context success
[INFO] create stream success
[INFO] dvp init resource success
[INFO] start to process picture: ./data/wood_rabbit_1024_1058_nv12.yuv
[INFO] call JpegE
[INFO] INFO end to destroy stream
[INFO] end to destroy context
[INFO] end to reset device is 0
```
After the executable file is executed successfully, a result file is generated in the result directory at the same level as the main file for later query. The result file includes the following:

- `dvpp_output_0`: output image generated after `persian_cat_1024_1536_283.jpg` is resized, cropped, or cropped and pasted to the target image.
- `dvpp_output_1`: output image generated after `wood_rabbit_1024_1061_330.jpg` is resized, cropped, or cropped and pasted to the target image.
- `model_output_0`: model inference result (a binary file) of `persian_cat_1024_1536_283.jpg`.
- `model_output_0.txt`: model inference result (a .txt file) of `persian_cat_1024_1536_283.jpg`.
- `model_output_1`: model inference result (a binary file) of `wood_rabbit_1024_1061_330.jpg`.
- `model_output_1.txt`: model inference result (a .txt file) of `wood_rabbit_1024_1061_330.jpg`.
- `jpege_output_0.jpg`: output image after `wood_rabbit_1024_1068_nv12.yuv` is encoded.

8.5 Image Classification Based on Caffe ResNet-50 (Including Video Decoding)

8.5.1 Sample Overview

Obtaining Sample

Obtain the code of sample `acl_vdec_resnet50` from the `/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gcc.x.x/accelib/sample/acl_execute_model` or `/home/HwHiAiUser/AscendMiniOs/accelib/sample/acl_execute_model` directory in the development environment.

Function Description

This sample shows how to classify images based on the Caffe ResNet-50 network (single input and single batch).

Convert the model file of the Caffe ResNet-50 network into an offline model adapted to the Ascend AI Processor (an *.om file). In the sample, load the .om file and decode a *.h265 video stream (containing only one frame) for 10 times to obtain 10 YUV420SP NV12 images, resize the 10 images, and infer their formats to obtain the inference results. Then, process the inference results and output the category IDs with the corresponding maximum confidences and the sum of the top 5 confidences.

During model conversion, you need to set the CSC parameters to convert YUV420SP images into RGB images to meet the input requirements of the model.
## Principles

The following table lists the key functions involved in this sample. For details about APIs, see See Also > ACL API Reference.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initialization</strong></td>
<td>- Call <code>aclInit</code> to initialize the ACL configuration.</td>
</tr>
<tr>
<td></td>
<td>- Call <code>aclFinalize</code> to deinitialize the ACL configuration.</td>
</tr>
<tr>
<td><strong>Device Management</strong></td>
<td>- Call <code>aclrtSetDevice</code> to specify the device for computation.</td>
</tr>
<tr>
<td></td>
<td>- Call <code>aclrtGetRunMode</code> to obtain the running mode of the Ascend AI Software Stack. The internal processing process varies according to the running mode.</td>
</tr>
<tr>
<td></td>
<td>- Call <code>aclrtResetDevice</code> to reset the current device and reclaim the resources on the device.</td>
</tr>
<tr>
<td><strong>Context Management</strong></td>
<td>- Call <code>aclrtCreateContext</code> to create a context.</td>
</tr>
<tr>
<td></td>
<td>- Call <code>aclrtDestroyContext</code> to destroy the context.</td>
</tr>
<tr>
<td><strong>Streams Management</strong></td>
<td>- Call <code>aclrtCreateStream</code> to create a stream.</td>
</tr>
<tr>
<td></td>
<td>- Call <code>aclrtDestroyStream</code> to destroy a stream.</td>
</tr>
<tr>
<td></td>
<td>- Call <code>aclrtSynchronizeStream</code> to block the host processing until all tasks in the specified stream are completed.</td>
</tr>
<tr>
<td><strong>Memory Management</strong></td>
<td>Call <code>aclrtMallocHost</code> to allocate memory on the host.</td>
</tr>
<tr>
<td></td>
<td>- Call <code>aclrtFreeHost</code> to free memory on the host.</td>
</tr>
<tr>
<td></td>
<td>- Call <code>aclrtMalloc</code> to allocate memory on the device.</td>
</tr>
<tr>
<td></td>
<td>- Call <code>aclrtFree</code> to free the memory on the device.</td>
</tr>
<tr>
<td></td>
<td>At data preprocessing, if you need to allocate the memory on the device to store the input or output data, call <code>acldvppMalloc</code> to allocate memory and call <code>acldvppFree</code> to free the memory.</td>
</tr>
<tr>
<td><strong>Data Movement</strong></td>
<td>Call <code>aclrtMemcpy</code> to:</td>
</tr>
<tr>
<td></td>
<td>- Copy data from the host to the device as the input data for decoding.</td>
</tr>
<tr>
<td></td>
<td>- Copy the inference result from the device to the host.</td>
</tr>
<tr>
<td><strong>Data Pre-processing</strong></td>
<td>- Video Decoding</td>
</tr>
<tr>
<td></td>
<td>Call <code>aclvdecSendFrame</code> to decode the video streams into YUV420SP images.</td>
</tr>
<tr>
<td></td>
<td>- Resizing:</td>
</tr>
<tr>
<td></td>
<td>Call <code>acldvppVpcResizeAsync</code> to resize a YUV420SP NV12 image to a 224 x 224 image.</td>
</tr>
<tr>
<td><strong>Model Inference</strong></td>
<td>- Call <code>aclmdlLoadFromFileWithMem</code> to load the model from the *.om file.</td>
</tr>
<tr>
<td></td>
<td>- Call <code>aclmdlExecute</code> to perform model inference.</td>
</tr>
<tr>
<td></td>
<td>Before inference, use the CSC parameters in the *.om file to convert a YUV420SP image into an RGB image.</td>
</tr>
<tr>
<td></td>
<td>- Call <code>aclmdlUnload</code> to unload the model.</td>
</tr>
</tbody>
</table>
Directory Structure

The code structure of sample acl_vdec_resnet50 is as follows:

- acl_vdec_resnet50
  ├── caffe_model
  │   ├── aipp.cfg  // Configuration file with CSC parameters, used for model conversion
  │   └── resnet50.prototxt  // Model file of the ResNet50 network
  ├── inc
  │   ├── dvpp_process.h  // Header file that declares functions related to model processing
  │   ├── model_process.h  // Header file that declares functions related to resource initialization/destruction
  │   └── sample_process.h  // Header file that declares functions related to model processing
  │       └── utils.h  // Header file that declares common functions (such as file reading function)
  │           └── vdec_process.h  // Header file that declares the video processing function
  └── src
    ├── acl.json  // Configuration file for system initialization
    ├── CMakeLists.txt  // Build script
    ├── dvpp_process.cpp  // Implementation file of functions related to data preprocessing
    │       └── main.cpp  // Implementation file of the main function and image classification function
    │           └── model_process.cpp  // Implementation file of model processing functions
    │               └── sample_process.cpp  // Implementation file of functions related to resource initialization/destruction
    │                   └── utils.cpp  // Implementation file of common functions (such as the file read function)
    │                       └── vdec_process.cpp  // Implementation file that declares the video processing function
    └── .project  // Project information file, including the project type, project description, and type of the target device
        └── CMakeLists.txt  // Build script that calls the CMakeLists file in the src directory

8.5.2 Building and Running Application on Host

This document uses HwHiAiUser as an example.

**Step 1** Convert your model.

1. Log in to the development environment as the HwHiAiUser user (running user).

2. Obtain the tool and set environment variables. For details, see [5.3 Model Conversion Using ATC](#).

3. Prepare data.

   Obtain the weight file (*.caffemodel) of the ResNet-50 network from the [https://gitee.com/HuaweiAscend/models/tree/master/computer_vision/classification/resnet50](https://gitee.com/HuaweiAscend/models/tree/master/computer_vision/classification/resnet50) and upload the file to the acl_vdec_resnet50/caffe_model directory in the development environment as the HwHiAiUser user (running user).

4. Convert the ResNet-50 network to an offline model adapted to the Ascend AI Processor (an *.om file). During model conversion, you need to set CSC parameters to convert YUV420SP images to RGB images.

   Go to the acl_vdec_resnet50 directory and run the following command:

   ```bash
   atc --model=caffe_model/resnet50.prototxt --weight=caffe_model/resnet50.caffemodel --framework=0 --output=model/resnet50_aipp --soc_version=Ascend310 --insert_op_conf=caffe_model/aipp.cfg
   ```

   **output**: The generated resnet50_aipp.om file is stored in the acl_vdec_resnet50/model directory. The default settings are recommended in...
the command. Otherwise, you need to change the value of `omModelPath` in `sample_process.cpp` before building the code.

```c
const char* omModelPath = "../model/resnet50_aipp.om";
```

For details about the parameters, see Restrictions and Parameters in ATC Tool Instructions.

**Step 2** Compile the code.

1. Prepare the input video stream.
   
   Obtain the input video stream file `vdec_h265_1frame_rabbit_1280x720.h265` from [https://gitee.com/HuaweiAscend/tools/tree/master/dvpp_sample_input_data](https://gitee.com/HuaweiAscend/tools/tree/master/dvpp_sample_input_data) and upload it to the acl_vdec_resnet50 directory of the development environment as the HwHiAiUser user (running user).

2. Log in to the development environment as the HwHiAiUser user (running user).

3. Set environment variables. The build script `src/CMakeLists.txt` builds code based on the header file path and library file path set by environment variables.

   The build script `CMakeLists.txt` is predefined with a default header file path `(/usr/local/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/include/)` and a default library file path `(/usr/local/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/lib64/stub/)`. If the `$DDK_PATH` and `$NPU_HOST_LIB` environment variables are not specified, the paths predefined in the build script `CMakeLists.txt` are used. If the `$DDK_PATH` and `$NPU_HOST_LIB` environment variables have been set, the build script `CMakeLists.txt` looks up the header files and library files that the build depends on in the paths specified by the environment variables.

   The following is an example of setting environment variables. Replace `/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x` with the actual ACLlib path.

   **Perform one of the following steps based on the actual situation:**

   If the OS architecture of the development environment is the same as that of the operating environment, run the following commands to import environment variables:

   ```bash
   export DDK_PATH=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x
   export NPU_HOST_LIB=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/lib64/stub
   ```

   When the operating environment is deployed on Atlas 500 AI edge stations, if the development environment uses the x86 architecture and the operating environment uses the ARM architecture, the ACLlib library of the ARM architecture development kit needs to be called during application compilation because the development kits of both the x86 and ARM architectures are deployed in the development environment. Therefore, you need to import the environment variables to the path of the ACLlib library in the ARM architecture.

   ```bash
   export DDK_PATH=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/arm64-linux_gcc7.3.0
   export NPU_HOST_LIB=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/arm64-linux_gcc7.3.0/acllib/lib64/stub
   ```
The .so library files in the /home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/lib64/stub directory are required to build the code logic based on the ACL APIs, without depending on any .so library files of other components (such as Driver).

After successful build, when you run an application on the host, the application can be linked to the .so library files in the /home/HwHiAiUser/Ascend/nmrt/latest/xxx-linux_gccx.x.x/acllib/lib64 directory on the host by configuring corresponding environmental variables. The application is automatically linked to the dependent .so library files of other components during running.

4. Go to the acl_vdec_resnet50 directory and create a directory for storing build outputs. For example, the directory created in this sample is build/intermediates/host.

   mkdir -p build/intermediates/host

5. Go to the build/intermediates/host directory and run the cmake command.

   ..../src indicates the directory where the CMakeLists.txt file is located. Replace it with the actual path.

   cd build/intermediates/host
   If the OS architecture of the development environment is the same as that of the operating environment, run the following command to perform compilation:
   cmake ../../../src -DCMAKE_CXX_COMPILER=g++ -DCMAKE_SKIP_RPATH=TRUE
   If the operating environment is deployed on an Atlas 500 AI edge station, the development environment uses the x86 architecture, and the operating environment uses the ARM architecture, run the following command to perform cross compilation:
   cmake ../../../src -DCMAKE_CXX_COMPILER=aarch64-linux-gnu-g++ -DCMAKE_SKIP_RPATH=TRUE

6. Run the make command to generate an executable file main in the acl_vdec_resnet50/out directory.

   make

Step 3 Run the application.

   NOTE

   • In the development and commissioning scenario, a compiled application is running in the development environment. Therefore, you need to import the following environment variables to the development environment:
     export LD_LIBRARY_PATH=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/lib64

     Then perform Step 3.5 to run the application.

   • In the operating scenario, you need to perform Step 3.1 to Step 3.5 to copy the application compiled in the development environment to the operating environment, and then run the application.

1. Upload the src, out, data, and model directories in the acl_vdec_resnet50 directory in the development environment to the same directory in the operating environment (host) as the HwHiAiUser user (execution user), for example, /home/HwHiAiUser/acl_vdec_resnet50.

2. Log in to the operating environment (host) as the HwHiAiUser user (running user).

3. Set the environment variable.
   The following is an example only. Replace the path with the actual one.
   export LD_LIBRARY_PATH=/home/HwHiAiUser/Ascend/nmrt/latest/xxx-linux_gccx.x.x/acllib/lib64:/usr/local/Ascend/add-ons

4. Go to the directory where the executable file main is located (for example, /home/HwHiAiUser/acl_vdec_resnet50/out) and grant the execute permission on the main file in the directory.
5. Go to the directory where the executable file `main` is located (for sample, `/home/HwHiAiUser/acl_vdec_resnet50/out`) and run the executable file.

```
./main
```

After the executable file is executed successfully, the following information is displayed:

```
......
[INFO] output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
[INFO] memcpy output data from device to host buffer success.
[INFO] create output file success, filename= ./result/model_output_0, size=4000
[INFO] start check result file: ./result/model_output_0
[INFO] check result success, file exist
[INFO] reslut file: [./result/model_output_0.txt]
[INFO] result: classType[331], top1[0.908203], top5[1.000015]
[INFO] Process dvpp success
[INFO] model execute success
[INFO] output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
[INFO] memcpy output data from device to host buffer success.
[INFO] create output file success, filename= ./result/model_output_1, size=4000
[INFO] start check result file: ./result/model_output_1
[INFO] check result success, file exist
[INFO] reslut file: [./result/model_output_1.txt]
[INFO] result: classType[688], top1[0.596680], top5[0.901611]
[INFO] Process dvpp success
[INFO] model execute success
[INFO] output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
[INFO] memcpy output data from device to host buffer success.
[INFO] create output file success, filename= ./result/model_output_2, size=4000
[INFO] start check result file: ./result/model_output_2
[INFO] check result success, file exist
[INFO] reslut file: [./result/model_output_2.txt]
[INFO] result: classType[331], top1[0.908203], top5[1.000015]
[INFO] Process dvpp success
[INFO] model execute success
[INFO] output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
[INFO] memcpy output data from device to host buffer success.
[INFO] create output file success, filename= ./result/model_output_3, size=4000
[INFO] start check result file: ./result/model_output_3
[INFO] check result success, file exist
[INFO] reslut file: [./result/model_output_3.txt]
[INFO] result: classType[331], top1[0.908203], top5[1.000015]
[INFO] Process dvpp success
[INFO] model execute success
[INFO] output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
[INFO] memcpy output data from device to host buffer success.
[INFO] create output file success, filename= ./result/model_output_4, size=4000
[INFO] start check result file: ./result/model_output_4
[INFO] check result success, file exist
[INFO] reslut file: [./result/model_output_4.txt]
[INFO] result: classType[331], top1[0.908203], top5[1.000015]
[INFO] Process dvpp success
[INFO] model execute success
[INFO] output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
[INFO] memcpy output data from device to host buffer success.
[INFO] create output file success, filename= ./result/model_output_5, size=4000
[INFO] start check result file: ./result/model_output_5
[INFO] check result success, file exist
[INFO] reslut file: [./result/model_output_5.txt]
[INFO] result: classType[331], top1[0.908203], top5[1.000015]
[INFO] Process dvpp success
[INFO] model execute success
[INFO] output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
[INFO] memcpy output data from device to host buffer success.
[INFO] create output file success, filename= ./result/model_output_6, size=4000
[INFO] start check result file: ./result/model_output_6
[INFO] check result success, file exist
[INFO] reslut file: [./result/model_output_6.txt]
```

INFO result: classType[331], top1[0.908203], top5[1.000015]
INFO Process dvpp success
INFO model execute success
INFO output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
INFO memcpy output data from device to host buffer success.
INFO create output file success, filename=./result/model_output_7, size=4000
INFO start check result file: ./result/model_output_7
INFO check result success, file exist
INFO reselut file: ./result/model_output_7.txt
INFO result: classType[331], top1[0.908203], top5[1.000015]
INFO Process dvpp success
INFO model execute success
INFO output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
INFO memcpy output data from device to host buffer success.
INFO create output file success, filename=./result/model_output_8, size=4000
INFO start check result file: ./result/model_output_8
INFO check result success, file exist
INFO reselut file: ./result/model_output_8.txt
INFO result: classType[331], top1[0.908203], top5[1.000015]
INFO Process dvpp success
INFO model execute success
INFO output[0] DataBuffer, buffer addr = 0x10100007c000, buffer size = 4000
INFO memcpy output data from device to host buffer success.
INFO create output file success, filename=./result/model_output_9, size=4000
INFO start check result file: ./result/model_output_9
INFO check result success, file exist
INFO reselut file: ./result/model_output_9.txt
INFO result: classType[331], top1[0.908203], top5[1.000015]
INFO unload model success, modelId is 1
INFO execute sample success
......

NOTE
- During the running of the executable file, after the VDEC successfully decodes the file and before model inference, an `outdir` directory is generated at the same level as the `main` file to store the decoded YUV420SP NV12 images. After the inference is complete, the `outdir` directory is cleared automatically.
- After the executable file is executed successfully, a result file is generated in the `result` directory at the same level as the `main` file for later query. The *.h265 video stream (containing only one frame) is decoded into a YUV420SP NV12 image. Perform resizing and model inference on this image for 10 times. 10 inference results are obtained.
  - `model_output_*`: the model inference result, a binary file
  - `model_output_*_.txt`: the model inference result, a TXT file

---End

8.6 Image Classification Based on Caffe ResNet-50
(Synchronous Inference, Excluding Data Pre-Processing)

8.6.1 Sample Overview

Obtaining Sample

Obtain the code of sample acl_resnet50 from the `/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acclib/sample/acl_execute_model` or /
home/HwHiAiUser/AscendMiniOs/aclib/sample/acl_execute_model directory in the development environment.

**Function Description**

This sample shows how to classify images based on the Caffe ResNet-50 network (single input and single batch).

Convert the model file of the Caffe ResNet-50 network into an offline model adapted to the Ascend AI Processor (an *.om file). In the sample, load the *.om file, decode, resize and infer two *.jpg images, obtain the inference results, process the inference results, and output the category IDs with the top 5 maximum confidences.

![Sample diagram](image)

**Figure 8-7** Sample diagram

**Principles**

The following table lists the key functions involved in this sample. For details about APIs, see See Also > ACL API Reference.

| Initializatio
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Call aclInit to initialize the ACL configuration.</td>
</tr>
<tr>
<td>• Call aclFinalize to deinitialize the ACL configuration.</td>
</tr>
</tbody>
</table>
## Device Management
- Call `aclrtSetDevice` to specify the device for computation.
- Call `aclrtGetRunMode` to obtain the running mode of the Ascend AI Software Stack. The internal processing process varies according to the running mode.
- Call `aclrtResetDevice` to reset the current device and reclaim the resources on the device.

## Context Management
- Call `aclrtCreateContext` to create a context.
- Call `aclrtDestroyContext` to destroy the context.

## Streams Management
- Call `aclrtCreateStream` to create a stream.
- Call `aclrtDestroyStream` to destroy a stream.

## Memory Management
- Call `aclrtMalloc` to allocate memory on the device.
- Call `aclrtFree` to free the memory on the device.

## Data Movement
Call `aclrtMemcpy` to:
- Copy data from the host to the device as the input data for inference.
- Copy the inference result from the device to the host.

## Model Inference
- Call `aclmdlLoadFromFileWithMem` to load the model from the *.om file.
- Call `aclmdlExecute` to perform model inference (synchronous API).
- Call `aclmdlUnload` to unload the model.

## Data Post-processing
Provides sample code to process the model inference result and display the category IDs with top 5 confidences on the device.

The sample provides a user-defined API `DumpModelOutputResult`, which is used to write the model inference result to a file (After the executable file is executed, the inference result file is generated in the directory where the executable file is located.) This API is not called by default. To call this API, you need to add the following code before the `OutputModelResult` API in `sample_process.cpp` in advance.

```c++
// print the top 5 confidence values with indexes.use function DumpModelOutputResult
// if want to dump output result to file in the current directory
processModel.DumpModelOutputResult();
processModel.OutputModelResult();
```

## Directory Structure
The code structure of the sample is as follows:

```
├acl_resnet50
│ ├── caffe_model
│ │   └── resnet50.prototxt //Model file of the ResNet50 network
```
Step 1  Convert your model.

1. Log in to the development environment as the HwHiAiUser user (running user).
2. Obtain the tool and set environment variables. For details, see 5.3 Model Conversion Using ATC.
3. Prepare data.
   Obtain the weight file (*.caffemodel) of the ResNet-50 network from the https://gitee.com/HuaweiAscend/models/tree/master/computer_vision/classification/resnet50 and upload the file to the acl_resnet50/caffe_model directory in the development environment as the HwHiAiUser user (running user).
4. Convert the ResNet-50 network to an offline model adapted to the Ascend AI Processor (an *.om file).
   Go to the acl_resnet50 directory and run the following command:
   ```bash
   atc --model=caffe_model/resnet50.prototxt --weight=caffe_model/resnet50.caffemodel --framework=0 --output=model/resnet50 --soc_version=Ascend310 --input_format=NCHW --input_fp16_nodes=data --output_type=FP32 --out_nodes=prob:0
   ```
   --output: The generated resnet50.om file is stored in the acl_resnet50/model directory. The default settings are recommended in the command.
   Otherwise, you need to change the value of omModelPath in sample_process.cpp before building the code.
   ```c
   const char* omModelPath = "./model/resnet50.om";
   ```
   For details about the parameters, see Restrictions and Parameters in ATC Tool Instructions.

Step 2  Compile the code.
1. Log in to the development environment as the HwHiAiUser user (running user).

2. Go to the acl_resnet50/data directory, run the transferPic.py script to convert the .jpg image to a .bin file, and resize the image from 1024 x 683 to 224 x 224. Two *.bin files are generated in the acl_resnet50/data directory.

   ```python
   python3.7.5 ../script/transferPic.py
   ```

   **NOTE**

   If the error message "ModuleNotFoundError: No module named 'PIL'" is displayed during script execution, the Pillow library does not exist. In this case, run the `pip3.7.5 install Pillow --user` command to install the Pillow library.

3. Set environment variables. The build script `src/CMakeLists.txt` builds code based on the header file path and library file path set by environment variables.

   The build script `CMakeLists.txt` is predefined with a default header file path (`/usr/local/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/include`) and a default library file path (`/usr/local/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/lib64/stub`).

   If the `${DDK_PATH}` and `${NPU_HOST_LIB}` environment variables are not specified, the paths predefined in the build script `CMakeLists.txt` are used. If the `${DDK_PATH}` and `${NPU_HOST_LIB}` environment variables have been set, the build script `CMakeLists.txt` looks up the header files and library files that the build depends on in the paths specified by the environment variables.

   The following is an example of setting environment variables. Replace `/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x` with the actual ACLlib path.

   **Perform one of the following steps based on the actual situation:**

   If the OS architecture of the development environment is the same as that of the operating environment, run the following commands to import environment variables:

   ```bash
   export DDK_PATH=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x
   export NPU_HOST_LIB=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/lib64/stub
   ```

   When the operating environment is deployed on Atlas 500 AI edge stations, if the development environment uses the x86 architecture and the operating environment uses the ARM architecture, the ACLlib library of the ARM architecture development kit needs to be called during application compilation because the development kits of both the x86 and ARM architectures are deployed in the development environment. Therefore, you need to import the environment variables to the path of the ACLlib library in the ARM architecture.

   ```bash
   export DDK_PATH=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/arm64-linux_gcc7.3.0
   export NPU_HOST_LIB=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/arm64-linux_gcc7.3.0/acllib/lib64/stub
   ```
The .so library files in the `/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/lib64/stub` directory are required to build the code logic based on the ACL APIs, without depending on any .so library files of other components (such as Driver).

After successful build, when you run an application on the host, the application can be linked to the .so library files in the `/home/HwHiAiUser/Ascend/nnrt/latest/xxx-linux_gccx.x.x/acllib/lib64` directory on the host by configuring corresponding environmental variables. The application is automatically linked to the dependent .so library files of other components during running.

4. Go to the `acl_resnet50` directory and create a directory for storing build outputs. For example, the directory created in this sample is `build/intermediates/host`.

```bash
cd build/intermediates/host
mkdir -p build/intermediates/host
```

5. Go to the `build/intermediates/host` directory and run the `cmake` command.

`../../../src` indicates the directory where the `CMakeLists.txt` file is located. Replace it with the actual path.

```bash
cd build/intermediates/host
```

If the OS architecture of the development environment is the same as that of the operating environment, run the following command to perform compilation:

```bash
cmake ../../../src -DCMAKE_CXX_COMPILER=g++ -DCMAKE_SKIP_RPATH=TRUE
```

If the operating environment is deployed on an Atlas 500 AI edge station, the development environment uses the x86 architecture, and the operating environment uses the ARM architecture, run the following command to perform cross compilation:

```bash
cmake ../../../src -DCMAKE_CXX_COMPILER=aarch64-linux-gnu-g++ -DCMAKE_SKIP_RPATH=TRUE
```

6. Run the `make` command to generate an executable file `main` in the `acl_resnet50/out` directory.

```bash
make
```

**Step 3** Run the application.

- In the development and commissioning scenario, a compiled application is running in the development environment. Therefore, you need to import the following environment variables to the development environment:

```bash
export LD_LIBRARY_PATH=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/lib64
```

Then perform **Step 3.5** to run the application.

- In the operating scenario, you need to perform **Step 3.1** to **Step 3.5** to copy the application compiled in the development environment to the operating environment, and then run the application.

1. Upload the `src`, `out`, `data`, and `model` directories in the `acl_resnet50` directory in the development environment to the same directory in the operating environment (host) as the `HwHiAiUser` user (running user), for sample, `/home/HwHiAiUser/ acl_resnet50`.

2. Log in to the operating environment (host) as the `HwHiAiUser` user (running user).

3. Set the environment variable.

   The following is an example only. Replace the path with the actual one.

   ```bash
   export LD_LIBRARY_PATH=/home/HwHiAiUser/Ascend/nnrt/latest/xxx-linux_gccx.x.x/acllib/lib64:/usr/local/Ascend/add-ons
   ```

4. Go to the directory where the executable file `main` is located (for example, `/home/HwHiAiUser/acl_resnet50/out`) and grant the execute permission on the `main` file in the directory.
chmod +x main

5. Go to the directory where the executable file main is located (for sample, /home/HwHiAiUser/acl_resnet50/out) and run the executable file.

./main

After the command is successfully executed, the following information is displayed.

```
INFO: acl init success
INFO: open device 0 success
INFO: create context success
INFO: create stream success
INFO: load model ./model/resnet50.om success
INFO: create model description success
INFO: create model output success
INFO: start to process file:../data/dog1_1024_683.bin
INFO: model execute success
INFO: top 1: index[161] value[0.900391]
INFO: top 2: index[164] value[0.638665]
INFO: top 3: index[163] value[0.019297]
INFO: top 4: index[162] value[0.016357]
INFO: top 5: index[167] value[0.012161]
INFO: output data success
INFO: start to process file:../data/dog2_1024_683.bin
INFO: model execute success
INFO: top 1: index[267] value[0.974609]
INFO: top 2: index[266] value[0.013062]
INFO: top 3: index[265] value[0.010017]
INFO: top 4: index[129] value[0.000335]
INFO: top 5: index[372] value[0.000179]
INFO: output data success
INFO: unload model success, modelId is 1
INFO: execute sample success
INFO: end to destroy stream
INFO: end to destroy context
INFO: end to reset device is 0
```

----End

8.7 Image Classification Based on Caffe ResNet-50 (Asynchronous Inference, Excluding Data Pre-Processing)

8.7.1 Sample Overview

Obtain Sample

Obtain the code of sample acl_resnet50_async from the /home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/sample/acl_execute_model or /home/HwHiAiUser/AscendMiniOs/acllib/sample/acl_execute_model directory in the development environment.
Function Description

This sample shows how to classify images based on the Caffe ResNet-50 network (single input and single batch).

Convert the model file of the Caffe ResNet-50 network into an offline model adapted to the Ascend AI Processor (an *.om file). In the sample, load the *.om file and perform \( n \) (an application parameter configured by the user) times of asynchronous inference on two *.jpg images. Then, obtain \( n \) inference results, process the inference results, and output the category ID with the top 1 confidence.

**Figure 8-8** Sample diagram

![Sample diagram]

Principles

The following table lists the key functions involved in this sample. For details about APIs, see See Also > ACL API Reference.

| Initialization | \- Call `aclInit` to initialize the ACL configuration.  
|               | \- Call `aclFinalize` to deinitialize the ACL configuration. |
| Device Management | Call **aclrtSetDevice** to specify the device for computation.  
Call **aclrtGetRunMode** to obtain the running mode of the Ascend AI Software Stack. The internal processing process varies according to the running mode.  
Call **aclrtResetDevice** to reset the current device and reclaim the resources on the device. |
| Context Management | Call **aclrtCreateContext** to create a context.  
Call **aclrtSetCurrentContext** to set the thread context.  
Call **aclrtDestroyContext** to destroy the context. |
| Streams Management | Call **aclrtCreateStream** to create a stream.  
Call **aclrtDestroyStream** to destroy a stream. |
| Memory Management | Call **aclrtMallocHost** to allocate memory on the host.  
Call **aclrtFreeHost** to free memory on the host.  
Call **aclrtMalloc** to allocate memory on the device.  
Call **aclrtFree** to free the memory on the device. |
| Data Movement | Call **aclrtMemcopy** to:  
- Copy data from the host to the device as the input data for inference.  
- Copy the inference result from the device to the host. |
| Model Inference | Call **aclmdlLoadFromFileWithMem** to load the model from the *.om file.  
Create a thread (for example, t1), call **aclrtProcessReport** in the thread function, and trigger the callback function (for example, **CallBackFunc** for processing the model inference result) after a specified time.  
Call **aclrtSubscribeReport** and specify the thread (t1) for processing the callback function (**CallBackFunc**) of the stream.  
Call **aclmdlExecuteAsync** to perform model inference (asynchronous API).  
Call **aclrtLaunchCallback** to add a callback function (**CallBackFunc**) to be executed on the host or device to the stream task queue.  
Call **aclrtSynchronizeStream** to block the application running until all tasks in the specified stream are complete.  
Call **aclrtUnSubscribeReport** to cancel the thread registration. The callback function (**CallBackFunc**) of the stream is not processed by the specified thread (t1).  
Call **aclmdlUnload** to unload the model after the model inference is complete. |
Data Post-processing

Provides sample code to process the model inference result and display the category IDs with top 5 confidences on the device.

The sample provides a user-defined API `DumpModelOutputResult`, which is used to write the model inference result to a file (After the executable file is executed, the inference result file is generated in the directory where the executable file is located.) This API is not called by default. To call this API, you need to add the following code before the `OutputModelResult` API in `sample_process.cpp` in advance.

```cpp
// print the top 5 confidence values with indexes.use function DumpModelOutputResult
// if want to dump output result to file in the current directory
processModel.DumpModelOutputResult();
processModel.OutputModelResult();
```

Directory Structure

The code structure of the sample is as follows:

- acl_resnet50_async
  - caffe_model
    - resnet50.prototxt //Model file of the ResNet50 network
  - data
    - dog1_1024_683.jpg //Test data
    - dog2_1024_683.jpg //Test data
  - inc
    - memory_pool.h //Header file that declares functions related to memory pool processing
    - model_process.h //Header file that declares functions related to model processing
    - sample_process.h //Header file that declares functions related to resource initialization/destruction
    - utils.h //Header file that declares common functions (such as file reading function)
  - script
    - transferPic.py //Convert a *.jpg image to a *.bin file and resize the image from 1024 x 683 to 224 x 224.
  - src
    - acl.json //Configuration file for system initialization
    - CMakeLists.txt //build script
    - main.cpp //Implementation file of the main function and image classification function
    - memory_pool.cpp //Implementation file of functions related to memory pool processing
    - model_process.cpp //Implementation file of model processing functions
    - sample_process.cpp //Implementation file of functions related to resource initialization/destruction
    - utils.cpp //Implementation file of common functions (such as the file read function)
  - .project //Project information file, including the project type, project description, and type of the target device
  - CMakeLists.txt //Build script that calls the CMakeLists file in the src directory

8.7.2 Building and Running Application on Host

This document uses HwHiAiUser as an example.

**Step 1** Convert your model.

1. Log in to the development environment as the HwHiAiUser user (running user).
2. Obtain the tool and set environment variables. For details, see 5.3 Model Conversion Using ATC.

3. Prepare data.
   Obtain the weight file (*.caffemodel) of the ResNet-50 network from the https://gitee.com/HuaweiAscend/models/tree/master/computer_vision/classification/resnet50 and upload the file to the acl_resnet50_async/caffe_model directory in the development environment as the HwHiAiUser user (running user).

4. Convert the ResNet-50 network to an offline model adapted to the Ascend AI Processor (an *.om file).
   Go to the acl_resnet50_async directory and run the following command:
   ```
   atc --model=caffe_model/resnet50.prototxt --weight=caffe_model/resnet50.caffemodel --framework=0 --output=model/resnet50 --soc_version=Ascend310 --input_format=NCHW --input_fp16_nodes=data --output_type=FP32 --out_nodes=prob:0
   --output: The generated resnet50.om file is stored in the acl_resnet50_async/model directory. The default settings are recommended in the command. Otherwise, you need to change the value of omModelPath in sample_process.cpp before building the code.
   ```

   For details about the parameters, see Restrictions and Parameters in ATC Tool Instructions.

Step 2 Build the code.

1. Log in to the development environment as the HwHiAiUser user (running user).

2. Go to the acl_resnet50_async/data directory, run the transferPic.py script to convert the .jpg image to a .bin file, and resize the image from 1024 x 683 to 224 x 224. Two *.bin files are generated in the acl_resnet50_async/data directory.

   ```python
   python3.7.5 ../script/transferPic.py
   ```

   **NOTE**
   If the error message "ModuleNotFoundError: No module named'PIL'" is displayed during script execution, the Pillow library does not exist. In this case, run the `pip3.7.5 install Pillow --user` command to install the Pillow library.

3. Set environment variables. The build script src/CMakeLists.txt builds code based on the header file path and library file path set by environment variables.

   The build script CMakeLists.txt is predefined with a default header file path (/usr/local/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/include/) and a default library file path (/usr/local/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/lib64/stub/).

   If the $DDK_PATH and $NPU_HOST_LIB environment variables are not specified, the paths predefined in the build script CMakeLists.txt are used. If the $DDK_PATH and $NPU_HOST_LIB environment variables have been set, the build script CMakeLists.txt looks up the header files and library files that the build depends on in the paths specified by the environment variables.

   The following is an example of setting environment variables. Replace `/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x` with the actual standard-form ACLlib installation path.

   Perform one of the following steps based on the actual situation:
If the OS architecture of the development environment is the same as that of the operating environment, run the following commands to import environment variables:

```
export DDK_PATH=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x
export NPU_HOST_LIB=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/lib64/stub
```

When the operating environment is deployed on Atlas 500 AI edge stations, if the development environment uses the x86 architecture and the operating environment uses the ARM architecture, the ACLlib library of the ARM architecture development kit needs to be called during application compilation because the development kits of both the x86 and ARM architectures are deployed in the development environment. Therefore, you need to import the environment variables to the path of the ACLlib library in the ARM architecture.

```
export DDK_PATH=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/arm64-linux_gcc7.3.0
export NPU_HOST_LIB=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/arm64-linux_gcc7.3.0/acllib/lib64/stub
```

**NOTE**

The .so library files in the `/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/lib64/stub` directory are required to build the code logic based on the ACL APIs, without depending on any .so library files of other components (such as Driver).

After successful build, when you run an application on the host, the application can be linked to the .so library files in the `/home/HwHiAiUser/Ascend/nnrt/latest/xxx-linux_gccx.x.x/acllib/lib64` directory on the host by configuring corresponding environmental variables. The application is automatically linked to the dependent .so library files of other components during running.

4. Go to the `acl_resnet50_async` directory and create a directory for storing build outputs. For example, the directory created in this sample is `build/intermediates/host`.

```
mkdir -p build/intermediates/host
```

5. Go to the `build/intermediates/host` directory and run the `cmake` command.

```
./../../src
```

Replace it with the actual path.

```
cd build/intermediates/host
```

If the OS architecture of the development environment is the same as that of the operating environment, run the following command to perform compilation:

```
cmake ../../../src -DCMAKE_CXX_COMPILER=g++ -DCMAKE_SKIP_RPATH=TRUE
```

If the operating environment is deployed on an Atlas 500 AI edge station, the development environment uses the x86 architecture, and the operating environment uses the ARM architecture, run the following command to perform cross compilation:

```
cmake ../../../src -DCMAKE_CXX_COMPILER=aarch64-linux-gnu-g++ -DCMAKE_SKIP_RPATH=TRUE
```

6. Run the `make` command to generate an executable file `main` in the `acl_resnet50_async/out` directory.

```
maked
```

**Step 3** Run the application.
In the development and commissioning scenario, a compiled application is running in the development environment. Therefore, you need to import the following environment variables to the development environment:

```bash
export LD_LIBRARY_PATH=/home/HwHiAiUser/Ascend/ascend-toolkit/latest/xxx-linux_gccx.x.x/acllib/lib64
```

Then perform **Step 3.5** to run the application.

In the operating scenario, you need to perform **Step 3.1** to **Step 3.5** to copy the application compiled in the development environment to the operating environment, and then run the application.

1. Upload the `src`, `out`, `data`, and `model` directories in the `acl_resnet50_async` directory in the development environment to the same directory in the operating environment (host) as the `HwHiAiUser` user (running user), for example, `/home/HwHiAiUser/acl_resnet50_async`.

2. Log in to the operating environment (host) as the `HwHiAiUser` user (running user).

3. Set the environment variable.

   The following is an example only. Replace the path with the actual one.

   ```bash
   export LD_LIBRARY_PATH=/home/HwHiAiUser/Ascend/nnrt/latest/xxx-linux_gccx.x.x/acllib/lib64:/usr/local/Ascend/add-ons
   ```

4. Go to the directory where the executable file `main` is located (for example, `/home/HwHiAiUser/acl_resnet50_async/out`) and grant the execute permission on the `main` file in the directory.

   ```bash
   chmod +x main
   ```

5. Go to the directory where the executable file `main` is located (for sample, `/home/HwHiAiUser/acl_resnet50_async/out`) and run the executable file.

   - Run the executable file without parameters:
     - By default, model asynchronous inference is performed for 100 times.
     - By default, the callback interval is 1, indicating that a callback task is issued after one time asynchronous inference.
     - By default, there are 100 memory blocks in a memory pool.

   - Run the executable file with parameters:
     - The first parameter indicates the number of model asynchronous inference times.
     - The second parameter indicates the interval for delivering the callback task. 0 indicates the callback task is not delivered. A non-zero value (for example, `m`) indicates that the callback task is delivered after `m` times of asynchronous inference.
     - The third parameter indicates the number of memory blocks in the memory pool. The argument must be greater than or equal to the number of model asynchronous inference times.

   ```bash
   ./main
   ```

   After the command is successfully executed, the following information is displayed.

   ```bash
   [INFO] ./main param1 param2 param3, param1 is execute model times(default 100), param2 is callback interval(default 1), param3 is memory pool size(default 100)
   ```
8.8 CompileDemo

8.8.1 Sample Principles

As we all know, most programmers start from writing a Hello World! small program. That is, the program prints Hello world! on the screen. This indicates that your programming journey begins. For Atlas programming, you can also start from writing Hello Davinci.

To help you quickly start an application development journey based on the AscendCL architecture on the Da Vinci chip, CompileDemo provides the methods of using CMake to compile projects in the AscendCL architecture and samples for initializing chips and creating and destroying contexts and streams. Initializing chips and creating and destroying the context and stream are basic operations for programming the Da Vinci chip by each Atlas application. In this basic operation, the CompileDemo sample implements the communication process in which the Hello Davinci message is written on the device and then sent to the host by the device. Figure 8-9 shows the sample process.

Figure 8-9 CompileDemo process
8.8.2 Code Implementation Parsing

Table 8-3 describes the functions called by the `main` function in the CompileDemo code. Figure 8-10 shows the `main` function calling process.

Table 8-3 Main functions of CompileDemo

<table>
<thead>
<tr>
<th>Function</th>
<th>Function Description</th>
<th>Parameter Description</th>
</tr>
</thead>
</table>
| InitialAscend  | Initializes a chip and create a context and stream.                                  | `int32_t deviceId`  
Device ID in which a context is to be created  
`aclrtContext &context`  
Pointer to the created context  
`aclrtStream &stream`  
Pointer to the created stream |
| aclrtMalloc    | ACL interface, which is used to apply for the memory on the device.                   | `void **devPtr`  
Pointer to the pointer to the allocated memory on the device  
`size_t size`  
Allocated memory size, in bytes  
aclrtMemMallocPolicy policy  
Memory allocation policy  
typedef enum aclrtMemMallocPolicy{   
ACL_MEM_MALLOC_HUGE_FIRST, // Prioritize the allocation of huge page memory. If the huge page memory is insufficient, normal page memory is allocated.   
ACL_MEM_MALLOC_HUGE_ONLY, // Allocate only huge page memory. If the huge page memory is insufficient, an error is returned.   
ACL_MEM_MALLOC_NORMAL_ONLY, // Allocate only normal page memory. } aclrtMemMallocPolicy; |
| aclrtMallocHost| ACL interface, which is used to apply for the memory on the host.                     | `void **hostPtr`  
Pointer to the allocated memory  
`size_t size`  
Allocated memory size, in bytes |
<table>
<thead>
<tr>
<th>Function</th>
<th>Function Description</th>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aclrtMemcpy</td>
<td>ACL interface, which is used to copy the memory on the host or device.</td>
<td>void *dst&lt;br&gt;Pointer to the destination memory address&lt;br&gt;size_t destMax&lt;br&gt;Maximum size of the destination memory address, in bytes&lt;br&gt;const void *src&lt;br&gt;Pointer to the source memory address&lt;br&gt;size_t count&lt;br&gt;Size of the copied memory, in bytes&lt;br&gt;aclrtMemcpyKind kind&lt;br&gt;Memory copy kind. This parameter is reserved and not supported currently.</td>
</tr>
<tr>
<td>DestoryAscend</td>
<td>Reset the chip and destroy the context and stream.</td>
<td>int32_t deviceID&lt;br&gt;ID of the device whose context is to be destroyed&lt;br&gt;aclrtContext &amp;context&lt;br&gt;Pointer to the destroyed context&lt;br&gt;aclrtStream &amp;stream&lt;br&gt;Pointer to the destroyed stream</td>
</tr>
</tbody>
</table>

**Figure 8-10** Main function calling process

### 8.8.3 Code Directory

The following describes the code directories and files of CompileDemo.

```
CompileDemo
├── build.sh                        // Build script
├── CMakeLists.txt                  // CMake build script
├── data                            // /Data storage directory
│   └── config                      // Directory for storing configuration files
```
8.8.4 Code Compilation

8.8.4.1 Initializing ACL

Principles

You must call the aclinit API to initialize the ACL. The configuration file is in JSON format. In this example, no configuration information is involved, and the configuration file is empty. For details about the configuration, see "Preparing Data > Preparing Dump Data for an Offline Model" in the Model Accuracy Analyzer Instructions (CLI).

Code Sample

After the API is called, add an exception handling branch and specify log printing of different levels (such as LogError and LogInfo).

The sample code is as follows. You can view the sample code in CompileDemo/main.cpp of the CompileDemo sample.

```cpp
#include "acl/acl.h"
//......
const char *aclConfigPath = ".\data\config\acl.json";
//Initialize basic ACL configurations.
//Create the directory for storing the configuration file of the character pointer variable, including the file name.
//The .. directory is relative to the directory of the executable file.
//For example, if the executable file is stored in the out directory, .. indicates the upper-level directory of the out directory.
APP_ERROR ret = aclInit(aclConfigPath);
//......
```

8.8.4.2 Runtime Resource Allocation

Principles

Allocate devices, contexts, and streams in sequence to support the execution of computing and management tasks.

The devices, contexts, and streams are described as follows:

- You can specify the device for calculation by calling aclrtSetDevice.
- When a context is created explicitly, aclrtCreateContext is called. In this case, aclrtDestroyContext needs to be called explicitly to release the context.
- A context explicitly created using aclrtCreateStream is destroyed using aclrtDestroyStream.
- A default context or stream created implicitly with the call to aclrtSetDevice has the following restrictions:
  - One device corresponds to one default context. The default context does not need to be destroyed by calling aclrtDestroyContext. To pass the default context to any API call, pass NULL directly.
- One device corresponds to one default stream. The default stream does not need to be destroyed by calling `aclrtDestroyStream`. To pass the default stream to any API call, pass **NULL** directly.

**Sample Code**

After the API is called, add an exception handling branch and specify log printing of different levels (such as **LogError** and **LogInfo**).

The sample code is as follows. You can view the sample code in the `InitialAscend` function of `CompileDemo/main.cpp` of the CompileDemo sample.

```c
#include "acl/acl.h"
#include <memory>

int32_t deviceId = 0;
aclrtContext context;
aclrtStream stream;

// 1. Specify the device for the computation.
ret = aclrtSetDevice(deviceID);
// 2. Explicitly create a context to manage the stream objects.
ret = aclrtCreateContext(&context, deviceID);
// 3. Explicitly create a stream.
// To reserve the execution order of asynchronous tasks.
ret = aclrtCreateStream(&stream);
```

8.8.4.3 "Hello DaVinci" Transmission and Backhaul

**Principles**

The **Hello Davinci** string is transmitted to the device and then to the host. In this sample, the process is as follows:

1. Create a character string.
   Create the **Hello Davinci** string in the host memory.

2. Apply for memory.
   Call the `aclrtMalloc` function to apply for the memory required for storing the **Hello Davinci** string on the device, and then call the `aclrtMallocHost` function on the host to apply for the memory of the same size.

3. Let the device communicate with the host.
   Transmit **Hello Davinci** to the device and then to the host to complete the communication between the device and host. Call the `aclrtMemcpy` function twice to implement two transmission processes.

**Sample Code**

After the API is called, add an exception handling branch and specify log printing of different levels (such as **LogError** and **LogInfo**).

The sample code is as follows. You can view the sample code in the `main` function of `CompileDemo/main.cpp` of the CompileDemo sample.

```c
#include "acl/acl.h"
#include <memory>

// 1. Create a character string.
// Create the string variable `str` to store "Hello Davinci".
```
const std::string str("Hello Davinci");
//2. Apply for memory.
//Create a pointer variable that points to the memory addresses of the device and host obtained by calling
//the memory application function.
//To prevent memory leakage, convert the allocated pointer to a smart pointer.
char *hostPtr, *devPtr;
ret = aclrtMalloc((void **)&devPtr, str.length(), ACL_MEM_MALLOC_NORMAL_ONLY);
std::shared_ptr<char> devBuf(devPtr, aclrtFree);
ret = aclrtMallocHost((void **)&hostPtr, str.length());
std::shared_ptr<char> hostBuf(hostPtr, aclrtFreeHost);
//3. Let the device communicate with the host.
//Call the memory copy function twice to complete the transmission and backhaul process.
ret = aclrtMemcpy(devBuf.get(), str.length(), str.c_str(), str.length(), ACL_MEMCPY_HOST_TO_DEVICE);
ret = aclrtMemcpy(hostBuf.get(), str.length(), str.c_str(), str.length(), ACL_MEMCPY_DEVICE_TO_HOST);

8.8.4.4 Runtime Resource Release

Principles

After all data on the host and device is processed, release the runtime resources, including the streams, contexts, and devices, in sequence.

The restrictions on releasing contexts and streams are as follows:

- The default context cannot be released using aclrtDestroyContext.
- The default stream cannot be released using aclrtDestroyStream.
- The default context and stream are automatically released with the reset of
device using aclrtResetDevice.

Sample Code

After the API is called, add an exception handling branch and specify log printing of
different levels (such as LogError and LogInfo).

The sample code is as follows. You can view the sample code in the
DestoryAscend function of CompileDemo/main.cpp of the CompileDemo
sample.

```
#include "acl/acl.h"
//......
//1. Release the stream.
ret = aclrtDestroyStream(stream);
//2. Release the context.
ret = aclrtDestroyContext(context);
//3. Release the device.
ret = aclrtResetDevice(deviceID);
//......
```

8.8.5 Code Compilation and Running

Compilation Tool

The advantage of CMake is that it is a cross-platform compilation tool. After you
write CMakelist.txt, CMake automatically generates the makefile or project file
that matches the environment.

For details about the cmake parameters, visit https://cmake.org/cmake/help/latest/guide/tutorial/index.html and query the required version.
Compilation and Running

This document uses HwHiAiUser as an example.

**Step 1** Build the code.

1. Log in to the development environment as the HwHiAiUser user.
2. Set environment variables. Replace \texttt{xxx-linux\_gccx.x.x} with the actual installation path.

**Perform one of the following steps based on the actual situation:**

If the OS architecture of the development environment is the same as that of the operating environment, run the following commands to import environment variables:

```bash
export ASCEND_HOME=/home/HwHiAiUser/Ascend
export ASCEND_VERSION=/ascend-toolkit/latest
export ARCH_PATTERN=/xxx-linux_gccx.x.x
export LD_LIBRARY_PATH=$ASCEND_HOME/$ASCEND_VERSION/$ARCH_PATTERN/acllib/lib64:$LD_LIBRARY_PATH
```

When the operating environment is deployed on Atlas 500 AI edge stations, if the development environment uses the x86 architecture and the operating environment uses the ARM architecture, the ACLlib library of the ARM architecture development kit needs to be called during application compilation because the development kits of both the x86 and ARM architectures are deployed in the development environment. Therefore, you need to import the environment variables to the path of the ACLlib library in the ARM architecture.

```bash
export ASCEND_HOME=/home/HwHiAiUser/Ascend
export ASCEND_VERSION=/ascend-toolkit/latest
export ARCH_PATTERN=/arm64-linux_gcc7.3.0
export LD_LIBRARY_PATH=$ASCEND_HOME/$ASCEND_VERSION/$ARCH_PATTERN/acllib/lib64:$LD_LIBRARY_PATH
```

3. Go to the root directory of the developed code folder and create the \texttt{CMakeList.txt} file. You can compile the file content as required. For details about the file compilation method and example, see \texttt{CMakelist Compilation Sample}.

4. Go to the CompileDemo directory and create a directory for storing build outputs. For example, the directory created in this sample is \texttt{build/intermediates/host}.

```bash
mkdir -p build/intermediates/host
```

5. Go to the \texttt{build/intermediates/host} directory and run the \texttt{cmake} command.

```bash
..../src
```

This indicates the directory where the \texttt{CMakeLists.txt} file is located. Replace it with the actual path.

6. Run the \texttt{make} command.

The generated executable file \texttt{main} is stored in the \texttt{dist} directory of the root directory.

7. Switch to the CompileDemo directory and run the following commands to copy the configuration file to the \texttt{dist} directory in the root directory:
Step 2 Run the code:

1. Log in to any directory on the host as the **HwHiAiUser** user.
2. Set the environment variable.
   
   The following is an example only. Replace the path with the actual one.
   
   ```bash
   export LD_LIBRARY_PATH=/home/HwHiAiUser/Ascend/nrrt/latest/xxx-linux_gccx.x.x/acllib/lib64:/usr/local/Ascend/add-ons
   ```
3. Upload the compiled code folder to the current directory.
4. Go to the directory where the executable file *main* is located, that is, the **dist** directory in the root directory of the code folder.
5. Run the `./main` command to run the executable file. The sample execution result is displayed.
   
   ```bash
   [Info ] [2020-07-10 22:53:32:995914][main.cpp GetDeviceID:49] The device ID is got successfully, deviceld = 0
   [Info ] [2020-07-10 22:53:32:996172][main.cpp InitAscend:67] The ACL is initialized successfully
   [Info ] [2020-07-10 22:53:33:315162][main.cpp InitAscend:74] The device is set successfully, deviceld = 0
   [Info ] [2020-07-10 22:53:33:316272][main.cpp InitAscend:81] The context is created successfully
   [Info ] [2020-07-10 22:53:33:316544][main.cpp InitAscend:88] The stream is created successfully
   [Info ] [2020-07-10 22:53:33:316584][main.cpp InitAscend:97] Running mode obtained successfully, mode = Host
   [Info ] [2020-07-10 22:53:33:324975][main.cpp DataInteraction:180] Send the "Hello Davinci" to device memory successfully
   [Info ] [2020-07-10 22:53:33:325096][main.cpp DataInteraction:188] Receive the "Hello Davinci" from device memory successfully
   [Info ] [2020-07-10 22:53:33:325252][main.cpp DestroyAscend:121] The stream is destroyed successfully
   [Info ] [2020-07-10 22:53:33:326250][main.cpp DestroyAscend:131] The context is destroyed successfully
   [Info ] [2020-07-10 22:53:33:442184][main.cpp DestroyAscend:138] Device reset successfully, deviceld = 0
   [Info ] [2020-07-10 22:53:33:442814][main.cpp DestroyAscend:145] The ACL is deinitialized successfully
   ```

   ----End

### CMakelist Compilation Sample

Understanding the compilation commands in the **CMakeList.txt** file helps you understand the code compilation process. **Table 8-4** describes the main compilation statements used in the **CMakeList.txt** file of the CompileDemo sample.

<table>
<thead>
<tr>
<th>Compilation Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmake_minimum_required</td>
<td>Specifies the earliest version of CMake required for compilation.</td>
</tr>
<tr>
<td>project</td>
<td>Specifies the project name.</td>
</tr>
<tr>
<td>add_definitions</td>
<td>Defines the macros required during compilation.</td>
</tr>
<tr>
<td>Compilation Statement</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| set                    | Assign values to the variables used during compilation.  
Example:  
set(CMAKE_SKIP_BUILD_RPATH True)  
The statement sets the **CMAKE_SKIP_BUILD_RPATH** variable to **True**. |
| get_filename_component | Obtains the file path.  
Example:  
get_filename_component(ASCEND_BASE_ABS_DIR ${FILE} ABSOLUTE)  
The statement assigns the absolute path of the file to **ASCEND_BASE_ABS_DIR**. |
| file                   | Obtains files recursively.  
Example:  
file(GLOBRECURSE src_file RELATIVE ${PROJECT_SOURCE_DIR} src/*.cpp)  
The statement indicates that files in the **src** directory and its subdirectories are obtained recursively and assigned to **src_file**. |
| add_compile_options    | Adds compilation options. |
| include_directories    | Includes the header files used by the project during compilation.  
Example:  
include_directories (/usr/local/Ascend/driver/include/dvpp/)  
The statement indicates that all header files in the **/usr/local/Ascend/driver/include/dvpp/** directory are included. |
| link_directories       | Adds the library dependency directories, including the static library and dynamic library.  
Example:  
link_directories(${SHARED_LIB_DIR})  
The statement indicates that the library files in the **SHARED_LIB_DIR** directory are linked during compilation. Note that **SHARED_LIB_DIR** is an absolute path. |
| add_executable         | Generates executable programs.  
Example:  
add_executable(main ${SOURCE_FILE})  
The statement compiles the contents of **SOURCE_FILE** into the executable file **main**. |
Compilation Statement | Description
--- | ---
target_link_libraries | Sets the name of the library file to be linked.
Example: target_link_libraries(main ascendcl acl_dvpp c_sec)
The statement indicates that the names of the library files to be linked by main are ascendcl, acl_dvpp, and c_sec.

The content of the CMakeList.txt file of the CompileDemo sample is as follows:
```cmake
# The earliest version of CMake on which compilation depends is 3.5.1.
cmake_minimum_required(VERSION 3.5.1)
# Set the project name to CompileDemo.
project(CompileDemo)
# Add the macro definition DENABLE_ACL_DVPP_INTERFACE for using the DVPP APIs.
add_definitions(-DENABLE_DVPP_INTERFACE)
# If the ASCEND_HOME environment variable is not set, the system generates a fatal log and stops the compilation.
if(NOT DEFINED ENV{ASCEND_HOME})
  message(FATAL_ERROR "please define environment variable:ASCEND_HOME")
endif()
# If the DRIVER_HOME environment variable is not set, set it to ASCEND_HOME. Otherwise, set it to the corresponding environment variable.
if(NOT DEFINED ENV{DRIVER_HOME})
  set(DRIVER_HOME $ENV{ASCEND_HOME})
else()
  set(DRIVER_HOME $ENV{DRIVER_HOME})
endif()
# If the ASCEND_VERSION environment variable is not set, the system generates a fatal log and stops the compilation.
if(NOT DEFINED ENV{ASCEND_VERSION})
  message(FATAL_ERROR "please define environment variable:ASCEND_VERSION")
endif()
# If the ARCH_PATTERN environment variable is not set, the system generates a fatal log and stops the compilation.
if(NOT DEFINED ENV{ARCH_PATTERN})
  message(FATAL_ERROR "please define environment variable:ARCH_PATTERN")
endif()
# Add compilation options.
add_compile_options(-std=c++11 -fPIE -fstack-protector-all -Werror)
# Do not use rpath during compilation.
set(CMAKE_SKIP_BUILD_RPATH True)
# Set the directory of the CMakeList.txt file to the root directory of the project.
set(PROJECT_SRC_ROOT ${CMAKE_CURRENT_LIST_DIR}/)
# Set the dist directory in the root directory of the project to the directory generated by the executable file.
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${PROJECT_SRC_ROOT}/dist)
# Find the Base directory in the relative directory of CMakeList.txt and assign the permission to ASCEND_BASE_DIR.
set(ASCEND_BASE_DIR ${CMAKE_CURRENT_SOURCE_DIR}/../ascendbase/src/Base)
# Find the absolute path of ASCEND_BASE_DIR and assign it to ASCEND_BASE_ABS_DIR.
get_filename_component(ASCEND_BASE_ABS_DIR $(CMKASE_CURRENT_LIST_DIR)/ ASCEND_BASE_DIR) ABSOLUTE)
## Obtain all source files in the ASCEND_BASE_ABS_DIR directory.
file(GLOB_RECURSE ASCEND_BASE_SRC_FILES $(ASCEND_BASE_ABS_DIR)/.*cpp)
# Set the directory of the header files to be contained.
## Set ALC INC_DIR to the directory where the header files of acllib are stored.
set(ALC_INC_DIR ${ENV{ASCEND_HOME}}/driver/kernel/ $ENV{ASCEND_HOME}/driver/include/dvpp/)
## Set CSEC INC_DIR to the directory where the header file of the secure function is stored.
set(CSEC_INC_DIR $ENV{ASCEND_HOME}/driver/kernel/ $ENV{ASCEND_HOME}/driver/include/)
## Set the header file directory of the ascend_hal function to HAL_INC_DIR.
set(HAL_INC_DIR $ENV{ASCEND_HOME}/driver/kernel/inc/driver/)
## Normalize the required header files to HEADER_FILE_DIR.
set(HEADER_FILE_DIR...)
```
\section*{8.9 InferClassification}

\subsection*{8.9.1 Sample Principles}

Let the computer "have" vision to identify the animal in the picture is cat or dog. Atlas developers can follow a standard development process to develop image processing AI applications.

1. Initialize resources.
2. Perform data pre-processing. The chip reads, decodes, and scales images in sequence.
4. Return the inference result to the host.

This section uses the \textbf{InferClassification} sample to describe how to classify images. \textbf{Figure 8-11} shows the main modules and working process of the \textbf{InferClassification} modules. \textbf{Table 8-5} describes the working process of the \textbf{InferClassification} modules.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure8-11.png}
\caption{Working process of the InferClassification module}
\end{figure}
<table>
<thead>
<tr>
<th>Module</th>
<th>Process Inside the Module</th>
</tr>
</thead>
</table>
| Resource Initialization | 1. Initialize ACL.  
2. Apply for runtime resources.  
3. Apply for DVPP resources.  
4. Apply for model inference resources.  
5. Initialize single-operator information. |
| Image decoding          | 1. Call `acldvppCreateChannel` to create a channel for image data processing.  
2. Call `acldvppMalloc` to apply for memory.  
3. Call `acldvppPixelFormat` to read the image from the folder.  
4. Call the asynchronous API `acldvppJpegDecodeAsync` for decoding.  
5. Call `acldvppFree` to free the input and output memory. |
| Image resizing          | 1. Call `acldvppCreateResizeConfig` to create image resize configuration.  
2. Call `acldvppMalloc` to apply for memory.  
3. Obtain the decoded image data.  
4. Call the `acldvppVpcResizeAsync` asynchronous API to resize the input image to the size of the output image.  
5. Call `acldvppFree` to free the input and output memory.  
6. Call `aacldvppDestroyResizeConfig` to destroy the picture scaling configuration.  
7. Call `acldvppDestroyChannel` to destroy the image data processing channel. |
| Image classification    | 1. Obtain the resized image data.  
2. Specify the data flow of the input model.  
3. Call `aclmdlExecute` to implement synchronous model inference based on the model ID returned during model loading.  
4. Obtain the model inference result. |
| Post-processing after classification | 1. Call `aclopCreateHandle` to create a handle for the operator.  
2. Call `aclopExecute` to asynchronously load and execute the operator based on the model ID returned during model loading.  
3. Call the `Cast` operator to convert the data type of the inference result from Float32 to Float16.  
4. Call the `ArgMaxD` operator to search for the category ID with the maximum confidence in the inference result. |
8.9.2 Code Implementation Parsing

The main implementation module of the InferClassification code consists of the main.cpp file, AclProcess.cpp file, and ascendbase common library file. Figure 8-12 shows the process of calling the main modules of the code. Table 8-6, Table 8-7, and Table 8-8 describe the module files and functions.

Figure 8-12 InferClassification modules and function calling

Table 8-6 Functions and parameters in the main.cpp file

<table>
<thead>
<tr>
<th>File Name</th>
<th>Function Name</th>
<th>Function Description</th>
<th>Parameter Description</th>
</tr>
</thead>
</table>
| main.cpp  | main          | Main function of the InferClassification sample. | int argc  
Abbreviation of argument count, indicating the number of parameters passed to the main function.  
char * argv[ ]  
Abbreviation of argument vector, indicating the parameter sequence or pointer passed to the main function. |
| Process   | Schedule the main module processes.  
ResourceInfo & resourceInfo  
Pointer to the resource object, including the chip ID, ACL configuration file path, and operator file storage path.  
std::string file  
Path for storing image files. |
<table>
<thead>
<tr>
<th>File Name</th>
<th>Function Name</th>
<th>Function Description</th>
<th>Parameter Description</th>
</tr>
</thead>
</table>
|            | ParseAndCheck Args | Parses and verifies parameters. | int argc  
The parameter description is the same as that of the main function.  
char * argv[]  
The parameter description is the same as that of the main function.  
CommandParser * options  
Pointer to the command parsing object. |
|            | ReadConfigFromFile | Reads configuration files. | ConfigParser & configData  
Pointer to the configuration parsing object.  
ResourceInfo & resourceInfo  
Pointer to a resource object. |
|            | ReadModelConfig | Reads the model configuration file. | ConfigParser & configData  
Pointer to the configuration parsing object.  
std::vector<ModelInfo> & modelInfos  
Container that bears a model object. |
Table 8-7 Functions in the AclProcess.cpp file and related common libraries

<table>
<thead>
<tr>
<th>Code File</th>
<th>Function</th>
<th>Function Description</th>
<th>Service Module</th>
<th>See Also</th>
<th>Function Description</th>
<th>Common Library Code Files</th>
<th>Common Library Called Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DVPPProcess.cpp</td>
<td>SingleOpProcess.h</td>
</tr>
<tr>
<td>InitResource</td>
<td></td>
<td>Resource initialization function that initializes the context, stream, DVPP, and operators.</td>
<td></td>
<td></td>
<td></td>
<td>ResourceManager.cpp</td>
<td>InitModule</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DvppProcess.cpp</td>
<td>InitOpCastResource</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ModelProcess.cpp</td>
<td>InitOpArgMaxResource</td>
</tr>
<tr>
<td>InitOpCastResource</td>
<td></td>
<td>Initializes the cast operator information.</td>
<td></td>
<td></td>
<td></td>
<td>SingleOpProcess.cpp</td>
<td>SetTypeName</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SetOpAttr</td>
<td>SetInputTensorNum</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SetInputTensor</td>
<td>SetOutputTensorNum</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SetOutputTensor</td>
<td>SetOutputTensorNum</td>
</tr>
<tr>
<td>Code File</td>
<td>Function</td>
<td>Function Description</td>
<td>Service Module</td>
<td>See Also</td>
<td>Function Description</td>
<td>Common Library Code Files</td>
<td>Common Library Called Functions</td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
<td>----------------------</td>
<td>----------------</td>
<td>----------</td>
<td>----------------------</td>
<td>---------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td></td>
<td>InitOpArgMaxResourse</td>
<td>Initializes the ArgMaxD operator information.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SetType Name SetOpAttr SetInputTensor Num SetInputTensor SetOutputTensor Num SetOutputTensor</td>
</tr>
<tr>
<td></td>
<td>Model Inference</td>
<td>Loads labels.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Model Inference</td>
<td>Model inference function, which inputs the preprocessed images into the model for inference.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DvppProcess.cpp ModelProcess.cpp</td>
</tr>
<tr>
<td>Code File</td>
<td>Function Name</td>
<td>Function Description</td>
<td>See Also</td>
<td>Common Library Code Files</td>
<td>Common Library Called Functions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>---------------</td>
<td>----------------------</td>
<td>----------</td>
<td>---------------------------</td>
<td>-------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Post-processing After Classification</td>
<td>CastOpInfer</td>
<td>Cast operator calling function, which converts the data type of the inference result from float32 to float16.</td>
<td>ModelProcess.cpp</td>
<td>getModelOutputData SetInputDataBuffer RunSingleOp</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ArgMaxOpInfer</td>
<td>Calls the <strong>ArgMaxD</strong> operator to search for the category ID with the maximum confidence in the inference result.</td>
<td></td>
<td>SignleOpProcess.cpp</td>
<td>GetOutputData GetOutputDataSize SetInputDataBuffer RunSingleOp</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WriteResult</td>
<td>Writes the image classification result to a file.</td>
<td></td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code File</td>
<td>Function</td>
<td>Function Description</td>
<td>Service Module</td>
<td>See Also</td>
<td>Function Description</td>
<td>Common Library Code Files</td>
<td>Common Library Called Functions</td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
<td>----------------------</td>
<td>----------------</td>
<td>----------</td>
<td>----------------------</td>
<td>---------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Resource Release</td>
<td></td>
<td>Release</td>
<td>Resource release function, which resets resources such as the dvpp, operator, and stream when the program ends.</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 8-8 Functions and parameters of the ascendbase common library file

<table>
<thead>
<tr>
<th>Common Library Code File</th>
<th>Usage</th>
<th>Function</th>
<th>Description</th>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DvppProcess.cpp</td>
<td>DVP processing</td>
<td>CreateDvp pChannel</td>
<td>Creates a channel for processing image data.</td>
<td>std::shared_ptr&lt;acldvppCh annelDesc&gt;&amp; dvppChannelDesc Smart pointer of the channel that carries image information.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DvppProcess</td>
<td>Schedules the DVPP module.</td>
<td>const aclrtStream &amp; stream Defines a stream.</td>
</tr>
<tr>
<td>DvppProcess.h</td>
<td>Image resizing header file</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>DvppJpegProcess.cpp</td>
<td>Image decoding</td>
<td>SetOutput Desc</td>
<td>Creates the description of the decoded output picture.</td>
<td>const uint32_t outputSize Defines the data type of the decoded image output as uint32_t.</td>
</tr>
<tr>
<td>Common Library Code File</td>
<td>Usage</td>
<td>Function</td>
<td>Description</td>
<td>Parameter Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------</td>
<td>-------------------</td>
<td>-------------</td>
<td>-----------------------</td>
</tr>
</tbody>
</table>
|                          |       | TransferImageH2D  | Copies image data from the host to the device. | const FileInfo& imageInfo
<p>|                          |       |                   |             | Define the pointer to the image information. |
|                          |       |                   |             | const uint32_t imageWidth |
|                          |       |                   |             | Define the image width. |
|                          |       |                   |             | The data type is uint32_t. |
|                          |       |                   |             | const uint32_t imageHeight |
|                          |       |                   |             | Define the image height. |
|                          |       |                   |             | The data type is uint32_t. |
|                          |       |                   |             | std::shared_ptr&lt;DvppDataInfoT&gt;&amp; jpegInput |
|                          |       |                   |             | Define the template pointer that bears the image data input information. |
|                          |       | ProcessFile       | Reads image information from a file. | const std::string&amp; imageFile |
|                          |       |                   |             | Define the image file path. |
|                          |       |                   |             | const acl::PixelFormat format |
|                          |       |                   |             | Define the image data type format. |
|                          |       |                   |             | const bool withSynchronize |
|                          |       |                   |             | Set the synchronization type to bool. |
|                          |       | ProcessHostData   | Reads the image information on the host. | const FileInfo&amp; imageInfo |
|                          |       |                   |             | Define the pointer to the image information. |
|                          |       |                   |             | const acl::PixelFormat format |
|                          |       |                   |             | Define the pixel format of the synchronization image. |
|                          |       |                   |             | const bool withSynchronize |
|                          |       |                   |             | Set the synchronization type to bool. |</p>
<table>
<thead>
<tr>
<th>Common Library Code File</th>
<th>Usage</th>
<th>Function</th>
<th>Description</th>
<th>Parameter Description</th>
</tr>
</thead>
</table>
|                          |        | ProcessDevData| Decodes images on the device. | const std::shared_ptr<DvppDataInfoT> jpegInput
|                          |        |               |             | Defines the template pointer that carries the image data information. |
|                          |        |               |             | const uint32_t outputSize |
|                          |        |               |             | Defines the decoding data type as uint32_t. |
|                          |        |               |             | const acldevppPixelFormat format |
|                          |        |               |             | Defines the decoding pixel format of a synchronous image. |
|                          |        |               |             | const bool withSynchronize |
|                          |        |               |             | Sets the synchronization type to bool. |

DvppJpegProcess.h

Image decode header file

N/A

N/A

N/A

DvppResizeProcess.cpp

Resizing

SetResizeParam

Sets the image resizing parameters.

uint32_t resizeWidth

Image resizing width type (uint32_t)

uint32_t resizeHeight

Image resizing height type (uint32_t)

acldevppPixelFormat format

Pixel format for image resizing

UpdateInput

Updates the description of the image decoding input.

N/A
<table>
<thead>
<tr>
<th>Common Library Code File</th>
<th>Usage</th>
<th>Function</th>
<th>Description</th>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>UpdateOutputDesc</td>
<td>Updates the description of the image decoding output.</td>
<td>N/A</td>
</tr>
</tbody>
</table>
|                          |       | CheckParams | Checks the image resizing parameters. | std::shared_ptr<DvppDataInfoT> inputData  
Defines the template that bears the input information about image scaling data.  
uint32_t _resizeWidth  
Image resizing width type (uint32_t)  
uint32_t _resizeHeight  
Image resizing height type (uint32_t)  
acldvppPixelFormat  
Pixel format for image resizing |
|                          |       | ProcessDevData | Resizes image on the device. | std::shared_ptr<DvppDataInfoT> inputData  
Defines the template that bears the input information about image scaling data.  
uint32_t _resizeWidth  
Image resizing width type (uint32_t)  
uint32_t _resizeHeight  
Image resizing height type (uint32_t)  
acldvppPixelFormat  
Pixel format for synchronous image resizing  
bool _withSynchronize  
Sets the synchronization type to bool. |
<table>
<thead>
<tr>
<th>Common Library Code File</th>
<th>Usage</th>
<th>Function</th>
<th>Description</th>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DvppResizeProcess.h</td>
<td>Image resizing header file</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ResourceManager.cpp</td>
<td>Resource management</td>
<td>InitResource</td>
<td>Resource initialization function</td>
<td>ResourceInfo&amp; resourceInfo Defines the pointer to the resource information.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LoadModels</td>
<td>Loads models and creates model descriptions based on model information.</td>
<td>std::vector&lt;ModelInfo&gt;&amp; modelInfos Defines the template pointer that carries model information.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LoadModeFromFileWithMem</td>
<td>Loads offline model data from a file.</td>
<td>ModelInfo&amp; modelInfo Defines the pointer to the model information.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CreateModelDesc</td>
<td>Describes the created model.</td>
<td>ModelInfo&amp; modelInfo Defines the pointer to the model information.</td>
</tr>
<tr>
<td>ResourceManager.h</td>
<td>Resource management header file</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ModelProcess.cpp</td>
<td>Model Inference</td>
<td>InitResource</td>
<td>Resource initialization function</td>
<td>N/A</td>
</tr>
<tr>
<td>Common Library Code File</td>
<td>Usage</td>
<td>Function</td>
<td>Description</td>
<td>Parameter Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SetInput</td>
<td>Creates a databuffer object and a datasetbuffer object for the input data.</td>
<td>const std::vector&lt;std::shared_ptr&lt;void&gt;&gt; &amp; inputDataBuffers</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Defines the pointer to the input data buffer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>const std::vector&lt;size_t&gt; &amp; buffersSize</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CreateOutput</td>
<td>Creates a dataset object and a databuffer object for the output data.</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ProcessHostBuf</td>
<td>Transmits input data from the host to the device.</td>
<td>const std::vector&lt;std::shared_ptr&lt;void&gt;&gt; &amp; inputHostBufs</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Defines the pointer of the hostbuffer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>const std::vector&lt;size_t&gt; &amp; buffersSize</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ProcessDevBuf</td>
<td>Performs model synchronization inference.</td>
<td>const std::vector&lt;std::shared_ptr&lt;void&gt;&gt; &amp; inputDevBufs</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Defines the pointer to inputDevBufs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>const std::vector&lt;size_t&gt; &amp; buffersSize</td>
</tr>
<tr>
<td></td>
<td></td>
<td>getModelOutputData</td>
<td>Obtains the output data, data size, and data address.</td>
<td>N/A</td>
</tr>
<tr>
<td>Common Library Code File</td>
<td>Usage</td>
<td>Function</td>
<td>Description</td>
<td>Parameter Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------</td>
<td>----------</td>
<td>-------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TransferH2D</td>
<td>Asynchronously replicates data from the host to the device.</td>
<td>const std::shared_ptr&lt;void&gt;&amp; inputHostBuff&lt;br&gt;Defines the smart pointer to inputHostBuff.&lt;br&gt;const unsigned int inBufferSize&lt;br&gt;Defines the buffer size.&lt;br&gt;std::shared_ptr&lt;void&gt;&amp; inputDeviceBuf&lt;br&gt;Defines the smart pointer to inputDeviceBuf.</td>
</tr>
<tr>
<td>ModelProcess.h</td>
<td></td>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td>Model inference header file</td>
<td></td>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td>SingleOperator.cpp</td>
<td></td>
<td>SetInputTensor</td>
<td>Creates input tensor description.</td>
<td>const std::vector&lt;Tensor&gt;&amp; tensors&lt;br&gt;Defines the smart pointer to tensors.</td>
</tr>
</tbody>
</table>
|                          |       | SetInputDataBuffer | Creates an input data buffer. | const std::vector<
|                          |       |           |             | std::shared_ptr<void>>& inputDataBuf<br>Defines the pointer to inputDataBuf.<br>const std::vector<
<p>|                          |       |           |             | size_t&gt; &amp;inputBufSize&lt;br&gt;Defines the pointer to inputBufSize. |
|                          |       | SetOutputTensor | Creates output tensor description and output data buffer. | const std::vector&lt;Tensor&gt;&amp; tensors&lt;br&gt;Defines the smart pointer to tensors. |</p>
<table>
<thead>
<tr>
<th>Common Library Code File</th>
<th>Usage</th>
<th>Function</th>
<th>Description</th>
<th>Parameter Description</th>
</tr>
</thead>
</table>
|                          |       | SetOpAttr| Sets specific properties of an operator based on the operator description type. | const std::shared_ptr<aclopAttr> &attr  
Defines the smart pointer to the aclopAttr type.  
const OpAttr &attrDesc  
Defines the operator attribute description. |
| SignleO pProces s.h      |       | N/A      | N/A         | N/A                   |
|                          |       | N/A      | N/A         | N/A                   |
| ConfigP arser.cpp        |       | ParseConfi g | Configures the parsing function. | const std::string &fileName  
Defines the configuration file path. |
|                          |       | GetStringV alue | Obtains the string value based on the key name. | const std::string &name  
Key name of the string type.  
std::string &value  
Value of the string type. |
|                          |       | getIntValu e | Obtains the int value based on the key name. | const std::string &name  
Key name of the string type.  
int &value  
Value of the string type. |
|                          |       | GetBoolVa lue | Obtains the bool value based on the key name. | const std::string &name  
Key name of the string type.  
bool &value  
Value of the string type. |
<table>
<thead>
<tr>
<th>Common Library Code File</th>
<th>Usage</th>
<th>Function</th>
<th>Description</th>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>GetFloatValue</td>
<td>Obtains the float value based on the key name.</td>
<td>const std::string &amp;name: Key name of the string type. float &amp;value: Value of the float type.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GetDoubleValue</td>
<td>Obtains the double value based on the key name.</td>
<td>const std::string &amp;name: Key name of the string type. double &amp;value: Value of the double type.</td>
</tr>
<tr>
<td>ConfigParser.h</td>
<td>N/A</td>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td>CommandParser.h</td>
<td>N/A</td>
<td></td>
<td></td>
<td>N/A</td>
</tr>
</tbody>
</table>

**NOTE**

You can download the sample and view the code file and function code involved in this sample based on the code file structure shown in 8.9.3 Code Directory to understand the code content in advance.

**8.9.3 Code Directory**

Directory structure of the ascendbase common library code file:

```markdown
ascendbase
   └── ci
   └── cmake
   └── CMakeLists.txt
   └── opensource
       └── output
```
Directory structure of the InferClassification sample code file:

```
InferClassification
├── AclProcess                                             // ACL operating management module, which manages the
│   ├── AclProcess.cpp
│   └── AclProcess.h
├── build.sh                                               // Build script
├── CMakeLists.txt                                         // CMake build script
├── CommandLine
│   └── CommandLine.h                                     // Command line
├── data                                                   // Data storage directory
│   ├── config                                            // Directory for storing configuration files
│   └── modelprocess                                      // Model processing module
├── modelprocess                                        // Model processing module
│   ├── modelprocess.cpp
│   └── modelprocess.h
├── modelmanager                                       // Model manager module
│   ├── modelmanager.cpp
│   └── modelmanager.h
├── vdecprocess                                          // VDEC processing module
│   ├── vdecprocess.cpp
│   └── vdecprocess.h
├── videocropprocess                                    // Video cropping processing module
│   └── videocropprocess.cpp
├── videocropmanager                                    // Video cropping manager module
│   └── videocropmanager.cpp
├── videocropmanager.h
├── videocropmanager.h
└── videocropmanager.cpp
```
8.9.4 Code Compilation

8.9.4.1 Main Program

Main Program Functions

The following table describes the functions called by the main function file `main.cpp` of `InferClassification`.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Sample Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParseAndCheckArgs</td>
<td>Parses and verifies parameters.</td>
<td>Parameter Parsing and Verification Function ParseAndCheckArgs</td>
</tr>
<tr>
<td>ReadModelConfig</td>
<td>Reads model configuration files.</td>
<td>Model Configuration File Reading Function ReadModelConfig</td>
</tr>
<tr>
<td>ReadConfigFromFile</td>
<td>Reads configuration files.</td>
<td>Configuration File Reading Function ReadConfigFromFile</td>
</tr>
<tr>
<td>Process</td>
<td>Schedules the main module processes.</td>
<td>Scheduling Function Process of the Main Module</td>
</tr>
<tr>
<td>main</td>
<td>Main function of the InferClassification sample.</td>
<td>Main Function main</td>
</tr>
</tbody>
</table>

**Parameter Parsing and Verification Function ParseAndCheckArgs**

main.cpp file: code example of the parameter parsing and verification function

```c++
#include "acl/acl.h"
//....

//Create the parameter parsing and verification function ParseAndCheckArgs.
// Parse and verify the input parameters of the image file.
APP_ERROR ParseAndCheckArgs(int argc, char *argv[], CommandParser *options)
{
    //Construct the parsing command.
    options->AddOption("-i", "/data/test.jpg", "Optional. Specify the input image, default: /data/test.jpg");
    options->ParseArgs(argc, argv);
    // Parse the image input file.
    std::string fileName = options->GetStringOption("-i");
```
Model Configuration File Reading Function ReadModelConfig

The following is a code example for reading the model configuration file:

```c++
#include "acl/acl.h"
//.....
// Create the ReadModelConfig function for reading the file model configuration file.
APP_ERROR ReadModelConfig(ConfigParser& configData, std::vector<ModelInfo>& modelInfos) {
    ModelInfo modelInfo;
    std::string tmp;
    // Obtain the model path from the configuration file.
    APP_ERROR ret = configData.GetStringValue("model_path", tmp);
    char absPath[PATH_MAX];
    // Obtain the absolute path of the model file.
    if (realpath(tmp.c_str(), absPath) == nullptr) {
        LogError << "Get the real path failed!";
        return APP_ERR_COMM_NO_EXIST;
    }
    // Check whether the absolute path of the model file is correct.
    int folderExist = access(absPath, R_OK);
    if (folderExist == -1) {
        LogError << "ModelPath " << absPath <<" doesn't exist or read failed!";
        return APP_ERR_COMM_NO_EXIST;
    }
    // Set the absolute path of the model file.
    modelInfo.modelPath = std::string(absPath);
    int tmpNum;
    // Obtain the model download mode from the configuration file.
    ret = configData.GetIntValue("model_load_method", tmpNum);
    // Set the model download method.
    modelInfo.method = (ModelLoadMethod)tmpNum;
    // Obtain the input width of the model from the configuration file.
    ret = configData.GetIntValue("model_width", tmpNum);
    modelInfo.modelWidth = tmpNum;
    // Obtain the input height of the model from the configuration file.
    ret = configData.GetIntValue("model_height", tmpNum);
    modelInfo.modelHeight = tmpNum;
    modelInfo.modelName = "resnet50";
    modelInfos.push_back(std::move(modelInfo));
    return APP_ERR_OK;
}
//.....
```

Configuration File Reading Function ReadConfigFromFile

The following is a code example for reading the configuration file:

```c++
#include "acl/acl.h"
//.....
// Create the ReadConfigFromFile function for reading the configuration file.
//Read the file configuration of the resnet50 model and singleOp operator.
APP_ERROR ReadConfigFromFile(ConfigParser& configData, ResourceInfo& resourceInfo) {
    std::string tmp;
    // Obtain the device ID used in the example from the configuration file.
    int deviceld;
    APP_ERROR ret = configData.GetIntValue("device_id", deviceld);
    // ID digital conversion
    // Check whether the device ID is correct.
    if (deviceld < 0) {
        LogError << "deviceld < 0, not valid, please check!";
    }
    //.....
```
Scheduling Function Process of the Main Module

The following is a code example of the module processing scheduling function Process:

```cpp
#include "acl/acl.h"

//......

APP_ERROR Process(ResourceInfo& resourceInfo, std::string file)
{
    std::shared_ptr<ResourceManager> instance = ResourceManager::GetInstance();
    int deviceId = *(resourceInfo.deviceIds.begin());
    //Call the GetContext function to obtain the context of the sample. For details about the function, see the GetContext Function in Runtime Management Resource Library File ResourceManager.cpp.
    aclrtContext context = instance->GetContext(deviceId);
    //Initialize an ACL sample module.
    AclProcess aclProcess(deviceId, resourceInfo.deviceResInfos[deviceId].modelInfos[0],
                           resourceInfo.singleOpFolderPath, context);
    APP_ERROR ret = aclProcess.InitResource();
    if (ret != APP_ERR_OK) {
        aclProcess.Release();
        return ret;
    }
    struct timeval begin = {0};
    struct timeval end = {0};
    gettimeofday(&begin, nullptr);
    ret = aclProcess.Process(file);
    if (ret != APP_ERR_OK) {
        aclProcess.Release();
        return ret;
    }
    gettimeofday(&end, nullptr);
    double costMs = SEC2MS * (end.tv_sec - begin.tv_sec) + (end.tv_usec - begin.tv_usec) / SEC2MS;
    double fps = SEC2MS / costMs;
    aclProcess.Release();
    return APP_ERR_OK;
}
//......
```

---

Atlas Intelligent Edge Solution
Application Software Development Guide

Scheduling Function Process of the Main Module

The following is a code example of the module processing scheduling function Process:

```cpp
#include "acl/acl.h"

//......

APP_ERROR Process(ResourceInfo& resourceInfo, std::string file)
{
    std::shared_ptr<ResourceManager> instance = ResourceManager::GetInstance();
    int deviceId = *(resourceInfo.deviceIds.begin());
    //Call the GetContext function to obtain the context of the sample. For details about the function, see the GetContext Function in Runtime Management Resource Library File ResourceManager.cpp.
    aclrtContext context = instance->GetContext(deviceId);
    //Initialize an ACL sample module.
    AclProcess aclProcess(deviceId, resourceInfo.deviceResInfos[deviceId].modelInfos[0],
                           resourceInfo.singleOpFolderPath, context);
    APP_ERROR ret = aclProcess.InitResource();
    if (ret != APP_ERR_OK) {
        aclProcess.Release();
        return ret;
    }
    struct timeval begin = {0};
    struct timeval end = {0};
    gettimeofday(&begin, nullptr);
    ret = aclProcess.Process(file);
    if (ret != APP_ERR_OK) {
        aclProcess.Release();
        return ret;
    }
    gettimeofday(&end, nullptr);
    double costMs = SEC2MS * (end.tv_sec - begin.tv_sec) + (end.tv_usec - begin.tv_usec) / SEC2MS;
    double fps = SEC2MS / costMs;
    aclProcess.Release();
    return APP_ERR_OK;
}
//......
```
Main Function main

The following is a code example of the main function in the main.cpp file:

```c++
#include "acl/acl.h"
//......
int main(int argc, char* argv[]) {
    AtlasAscendLog::Log::LogInfoOn();
    CommandParser options;
    //Parse and check parameters.
    APP_ERROR ret = ParseAndCheckArgs(argc, argv, &options);
    if (ret != APP_ERR_OK) {
        return ret;
    }
    std::string fileName = options.GetStringOption("-i");
    // Obtain the absolute path of the input file.
    char absPath[PATH_MAX];
    if (realpath(fileName.c_str(), absPath) == nullptr) {
        LogError << "Get the real path failed!";
        return APP_ERR_COMM_NO_EXIST;
    }
    std::string path(argv[0], argv[0] + strlen(argv[0]));
    ChangeDir(path.c_str());
    ConfigParser config;
    //Initialize the configuration parsing module.
    if (config.ParseConfig("../data/config/setup.config") != APP_ERR_OK) {
        return APP_ERR_COMM_INIT_FAIL;
    }
    ResourceInfo resourceInfo;
    //Path of the ACL configuration file
    resourceInfo.aclConfigPath = "../data/config/acl.json";
    // Read configuration parameters from the file.
    ret = ReadConfigFromFile(config, resourceInfo);
    //Obtain the ACL initialization and running management resources of the sample. For details about the code, see Runtime Management Resource Library File ResourceManager.cpp.
    std::shared_ptr<ResourceManager> instance = ResourceManager::GetInstance();
    //Call the InitResource function to obtain the sample file configuration. For details, see the InitResource function in Runtime Management Resource Library File ResourceManager.cpp.
    ret = instance->InitResource(resourceInfo);
    if (ret != APP_ERR_OK) {
        return -1;
    }
    std::string file = std::string(absPath);
    ret = Process(resourceInfo, file);
    if (ret != APP_ERR_OK) {
        instance->Release();
        return -1;
    }
    // Release sample resources.
    instance->Release();
    return 0;
}
```

8.9.4.2 Resource Initialization

8.9.4.2.1 Resource Initialization Functions

The InferClassification sample uses the following resources: ACL, runtime management resources, DVPP, model inference resources, and operators. For details about the resource initialization principles, see 5.4.1 Initializing Resources.

The ascendbase common function library encapsulates common classes and functions for system resource initialization. You only need to invoke the ascendbase common function library during resource initialization.
The resource initialization of the **InferClassification** sample is implemented using the `AclProcess.cpp` file. **Table 8-9** describes the resource initialization functions in the `AclProcess.cpp` file.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Sample Code</th>
<th>Common Library File Functions</th>
<th>Library File Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>InitResource</td>
<td>Resource initialization function</td>
<td><strong>8.9.4.2.2 Resource Initialization Functions</strong></td>
<td>Initializes ACL and operation management resources.</td>
<td><strong>Runtime Management Resource Library File ResourceManager.cpp</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Initializes the DVPP.</td>
<td><strong>DVPP Library File DvppProcess.cpp</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Initializes model inference resources.</td>
<td><strong>Model Inference Library File ModelProcess.cpp</strong></td>
</tr>
<tr>
<td>InitModule</td>
<td>AclProcess initialization module function</td>
<td><strong>8.9.4.2.3 Initialization Module Functions</strong></td>
<td>Creates objects such as DVPP encoding and DVPP decoding objects.</td>
<td>For details, see the class definition in the common library code.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>InitOpArgMaxResource</td>
<td>Initializes the ArgMax operator information.</td>
<td><strong>Single-operator Information Initialization Function InitOpArgMaxResource</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**8.9.4.2.2 Resource Initialization Functions**

**Resource Initialization Function InitResource**

The following is a code sample of initializing resources using the `AclProcess.cpp` file:

```
#include "acl/acl.h"
//......
```
Create a resource initialization function.

```c
APP_ERROR AclProcess::InitResource()
{
    // Set the current context of the thread.
    APP_ERROR ret = aclrtSetCurrentContext(context_);
    if (ret != APP_ERR_OK) {
        LogError << "acl Get context failed";
        return ret;
    }

    // Create a stream.
    ret = aclrtCreateStream(&stream_);
    if (ret != APP_ERR_OK) {
        LogError << "acl create stream failed!";
        return ret;
    }

    // Initialize the DVPP module.
    if (InitModule() != APP_ERR_OK) {
        return APP_ERR_COMM_INIT_FAIL;
    }

    // Initialize the Cast operator information module.
    if (InitOpCastResource() != APP_ERR_OK) {
        return APP_ERR_COMM_INIT_FAIL;
    }

    // Initialize the ArgMax operator information module.
    if (InitOpArgMaxResource() != APP_ERR_OK) {
        return APP_ERR_COMM_INIT_FAIL;
    }

    return APP_ERR_OK;
}
```

---

**Runtime Management Resource Library File ResourceManager.cpp**

The following is a code sample of initializing the ACL and applying for running management resources using the `ResourceManager.cpp` file:

```c
#include "acl/acl.h"
//......

// Create the InitResource function.
APP_ERROR ResourceManager::InitResource(ResourceInfo& resourceInfo)
{
    std::string aclConfigPath = resourceInfo.aclConfigPath;
    APP_ERROR ret = ExistFile(aclConfigPath);

    // Initialize the ACL.
    ret = aclInit(aclConfigPath.c_str());
    std::copy(resourceInfo.deviceIds.begin(), resourceInfo.deviceIds.end(), std::back_inserter(deviceIds_));
    // Open the device and configure a context for each chip.
    for (int i = 0; i < deviceIds_.size(); i++) {
        deviceIdMap_[deviceIds_[i]] = i;
        ret = aclrtSetDevice(deviceIds_[i]);
        aclrtContext context;
        // Explicitly create a context.
        ret = aclrtCreateContext(&context, deviceIds_[i]);
        if (ret != APP_ERR_OK)
            contexts_.push_back(context);
        if (resourceInfo.deviceResInfos.find(deviceIds_[i]) != resourceInfo.deviceResInfos.end()) {
            ret = LoadModels(resourceInfo.deviceResInfos[deviceIds_[i]].modelInfos);
            if (ret != APP_ERR_OK)
                contexts_[i].push_back(context);
            if (resourceInfo.deviceResInfos.find(deviceIds_[i]).modelInfos.end()) {
                ret = LoadModels(std::vector<ModelInfo>& modelInfos);
                if (ret != APP_ERR_OK)
                    contexts_[i].push_back(context);
            }
        }
    }

    // Create the LoadModels function.
    APP_ERROR ResourceManager::LoadModels(std::vector<ModelInfo>& modelInfos)
    {
        // Download all models.
        for (auto it = modelInfos.begin(); it != modelInfos.end(); it++) {
            //......
        }
    }
```
APP_ERROR ret = LoadModelFromFileWithMem(*it);
    // Create the description pointer to the download model.
    ret = CreateModelDesc(*it);
}
return APP_ERR_OK;
}

// Create the offline model download function LoadModelFromFileWithMem to download the resnet50 offline model.
APP_ERROR ResourceManager::LoadModelFromFileWithMem(ModelInfo& modelInfo)
{
    size_t modelMemSize;
    size_t modelWeightSize;
    // Obtain the weight memory size and working memory size required for model execution.
    APP_ERROR ret = aclmdlQuerySize(modelInfo.modelPath.c_str(), &modelMemSize, &modelWeightSize);
    // Allocate for the working memory for model execution on the device based on the working memory size.
    void* modelMemPtr = nullptr;
    ret = aclrtMalloc(&modelMemPtr, modelMemSize, ACL_MEM_MALLOC_NORMAL_ONLY);
    if (modelMemPtr != nullptr) {
        aclrtFree(modelMemPtr);
        modelMemPtr = nullptr;
    }
    return ret;
}

std::shared_ptr<void> modelMemShared(modelMemPtr, aclrtFree);
std::unordered_map<std::string, std::shared_ptr<void>> modelMemMap;
modelMemMap_.push_back(std::move(modelMemMap));
modelMemMap_[modelInfo.modelName] = modelMemShared;
// Allocate the weight memory for executing the model on the device based on the working memory size.
void* modelWeightPtr = nullptr;
ret = aclrtMalloc(&modelWeightPtr, modelWeightSize, ACL_MEM_MALLOC_NORMAL_ONLY);
if (ret != APP_ERR_OK) {
    LogError << "Malloc buffer for weight failed, require size is " << modelWeightSize;
    if (modelWeightPtr != nullptr) {
        aclrtFree(modelWeightPtr);
        modelWeightPtr = nullptr;
    }
    return ret;
}

std::shared_ptr<void> modelWeightShared(modelWeightPtr, aclrtFree);
std::unordered_map<std::string, std::shared_ptr<void>> modelWeightMap;
modelWeightMap_.push_back(std::move(modelWeightMap));
modelWeightMap_[modelInfo.modelName] = modelWeightShared;
// Load the offline model file (adapting to the offline model of the Ascend AI processor) and return the ID of the identified model.
ret = aclmdlLoadFromFileWithMem(modelInfo.modelPath.c_str(), &modelId, modelMemPtr,
modelMemSize,
    modelWeightPtr, modelWeightSize);
    std::unordered_map<std::string, int> modelIdMap;
    modelIdMap_.push_back(std::move(modelIdMap));
    modelIdMap_[modelInfo.modelName] = modelId;
}

// Create a model description object CreateModelDesc function to obtain the model description.
APP_ERROR ResourceManager::CreateModelDesc(ModelInfo& modelInfo)
{
    std::unordered_map<std::string, std::shared_ptr<aclmdlDesc>> modelDescMap;
    modelDescMap_.push_back(std::move(modelDescMap));
    int index = modelDescMap_.size() - 1;
    // Create a model description object pointer.
    aclmdlDesc* modelDesc = aclmdlCreateDesc();
    std::shared_ptr<aclmdlDesc> modelDescShared(modelDesc, aclmdlDestroyDesc);
    // Set the model description pointer.
    modelDescMap_[index][modelInfo.modelName] = modelDescShared;
    // Obtain the model description based on the model ID.
    APP_ERROR ret = aclmdlGetDesc(modelDesc, modelIdMap_[index][modelInfo.modelName]);
}

// Create a context.
aclrtContext ResourceManager::GetContext(int deviceId)
return contexts_[deviceIdMap_[deviceId]];
}
// Create the GetModelId function to obtain the ID of the loaded model.
APP_ERROR ResourceManager::GetModelId(int deviceId, std::string& modelName, uint32_t& modelId)
{
    if (modelIdMap_[deviceIdMap_[deviceId]].find(modelName) ==
        modelIdMap_[deviceIdMap_[deviceId]].end()) {
        return APP_ERR_INFER_FIND_MODEL_ID_FAIL;
    }
    modelId = modelIdMap_[deviceIdMap_[deviceId]][modelName];
    return APP_ERR_OK;
}
// Create the GetModelDesc function to obtain the description of the loaded model.
APP_ERROR ResourceManager::GetModelDesc(int deviceId, std::string& modelName,
std::shared_ptr<aclmdlDesc>& modelDesc)
{
    if (modelDescMap_[deviceIdMap_[deviceId]].find(modelName) ==
        modelDescMap_[deviceIdMap_[deviceId]].end()) {
        return APP_ERR_INFER_FIND_MODEL_DESC_FAIL;
    }
    modelDesc = modelDescMap_[deviceIdMap_[deviceId]][modelName];
    return APP_ERR_OK;
}
// Create the GetModelMem function to obtain the allocation information of the loaded model.
APP_ERROR ResourceManager::GetModelMem(int deviceId, std::string& modelName,
std::shared_ptr<void>& modelMem)
{
    if (modelMemMap_[deviceIdMap_[deviceId]].find(modelName) ==
        modelMemMap_[deviceIdMap_[deviceId]].end()) {
        return APP_ERR_INFER_FIND_MODEL_MEM_FAIL;
    }
    modelMem = modelMemMap_[deviceIdMap_[deviceId]][modelName];
    return APP_ERR_OK;
}
// Create the GetModelWeight function to obtain the weight of the loaded model.
APP_ERROR ResourceManager::GetModelWeight(int deviceId, std::string& modelName,
std::shared_ptr<void>& modelWeight)
{
    if (modelWeightMap_[deviceIdMap_[deviceId]].find(modelName) ==
        modelWeightMap_[deviceIdMap_[deviceId]].end()) {
        return APP_ERR_INFER_FIND_MODEL_WEIGHT_FAIL;
    }
    modelWeight = modelWeightMap_[deviceIdMap_[deviceId]][modelName];
    return APP_ERR_OK;
}
//......

DvppProcess.h Common Library File Code

The following is a code sample of creating the declaration alias DvppDataInfo
using the library file DvppProcess.h:

```cpp
#include "acl/acl.h"

//......
//Create the declaration alias DvppDataInfo of the image data description information of the DVPP modules.
typedef struct DvppDataInfo {
    std::shared_ptr<void> data;
    uint32_t lenOfByte;
    uint32_t width;
    uint32_t height;
    uint32_t widthStride;
    uint32_t heightStride;
    acldvppPixelFormat format;
} DvppDataInfoT;
//......
```
**DVPP Library File DvppProcess.cpp**

The following is a code sample of applying for model inference data preprocessing resources using the library file `DvppProcess.cpp`:

```cpp
#include "acl/acl.h"

//......
// Calls the CreateDvppChannel function for creating an image data processing channel.
APP_ERROR CreateDvppChannel(std::shared_ptr<acldvppChannelDesc>& dvppChannelDesc)
{
    dvppChannelDesc.reset(acldvppCreateChannelDesc(), DestroyChannelDesc);
    if (dvppChannelDesc.get() == nullptr) {
        LogError << "acldvppCreateChannelDesc failed!";
        return APP_ERR_COMM_INVALID_POINTER;
    }

    // Description of the channel for processing image data.
    APP_ERROR ret = acldvppCreateChannel(dvppChannelDesc.get());
    if (ret != APP_ERR_OK) {
        LogError << "acldvppCreateChannelAsync failed, ret = " << ret;
        return ret;
    }
    return APP_ERR_OK;
}

DvppProcess::DvppProcess(const aclrtStream& stream)
: stream_(stream), dvppChannelDesc_(nullptr), input_(nullptr), output_(nullptr)
{
    // Create an data processing channel in asynchronous mode.
    if (dvppChannelDesc_ == nullptr) {
        APP_ERROR ret = CreateDvppChannel(dvppChannelDesc_);
        if (ret != APP_ERR_OK) {
            LogError << "acldvppCreateChannel failed, ret = " << ret;
        }
    }
}

//......
```

**Model Inference Library File ModelProcess.cpp**

The following is a code sample of applying for model inference resources by using the model inference library file `ModelProcess.cpp`:

```cpp
#include "acl/acl.h"

//......
// Model resource initialization function
APP_ERROR ModelProcess::InitResource()
{
    // Create a stream pointer.
    APP_ERROR ret = aclrtCreateStream(&stream_);
    if (ret != APP_ERR_OK) {
        LogError << "Create stream failed";
        return ret;
    }

    std::shared_ptr<ResourceManager> instance = ResourceManager::GetInstance();
    //Call the GetModelId function to obtain the ID of the loaded model. For details about the function, see the clmdlDesc function in the library file.
    ret = instance->GetModelId(deviceId_, modelName_, modelId_);
    if (ret != APP_ERR_OK) {
        LogError << "GetModelId of " << modelName_ << " failed, please check the validity of modelName!";
        return ret;
    }

    //Call the aclmdlDesc function to obtain the type data of the model based on the ID of the loaded model. For details about how to call the function, see the GetModelDes function in Runtime Management Resource Library File ResourceManager.cpp.
    ret = instance->GetModelDesc(deviceId_, modelName_, modelDesc_);
    if (ret != APP_ERR_OK) {
        LogError << "GetModelDesc of " << modelName_ << " failed, please check the validity of";
    }
}
```
modelName!";
    return ret;
    
    // Create an output interface.
    ret = CreateOutput();
    if (ret != APP_ERR_OK) {
        LogError << "Create model output failed!";
        return ret;
    }
    initFlag = true;
    return APP_ERR_OK;
}

APP_ERROR ModelProcess::SetInput(const std::vector<std::shared_ptr<void>>& inputDataBuffers,
                                 const std::vector<size_t>& buffersSize)
{
    // Create data of the aclModelDataset type to describe the input of model inference.
    aclmdlDataset* dataSet = aclmdlCreateDataset();
    if (dataSet == nullptr) {
        LogError << "Create dataset for input failed!";
        return APP_ERR_COMM_INVALID_POINTER;
    }
    inputDataSet_.reset(dataSet, DestroyInput);
    for (int i = 0; i < inputDataBuffers.size(); i++) {
        // Obtain the input memory address for describing model inference.
        aclDataBuffer* inputData = aclCreateDataBuffer(inputDataBuffers[i].get(), buffersSize[i]);
        if (inputData == nullptr) {
            LogError << "Create data buffer failed!";
            return APP_ERR_COMM_INVALID_POINTER;
        }
        // Obtain the size of the input memory for model inference.
        APP_ERROR ret = aclmdlAddDatasetBuffer(inputDataSet_.get(), inputData);
        if (ret != APP_ERR_OK) {
            if (inputData != nullptr) {
                aclDestroyDataBuffer(inputData);
                inputData = nullptr;
            }
            return ret;
        }
    }
    return APP_ERR_OK;
}

APP_ERROR ModelProcess::CreateOutput()
{
    // Create data of the aclModelDataset type to describe the output of model inference.
    outputDataSet_.reset(aclmdlCreateDataset(), DestroyOutput);
    if (outputDataSet_ == nullptr) {
        LogError << "Create dataset output failed!";
        return APP_ERR_COMM_INVALID_POINTER;
    }
    // Obtain the number of output models.
    size_t outputSize = aclmdlGetNumOutputs(modelDesc_.get());
    // Allocate memory for each output with the for loop and add each output to the data of type aclModelDataset.
    for (size_t i = 0; i < outputSize; i++) {
        size_t bufferSize = aclmdlGetOutputSizeByIndex(modelDesc_.get(), i);
        void* outputBuffer = nullptr;
        APP_ERROR ret = aclrtMalloc(&outputBuffer, bufferSize, ACL_MEM_MALLOC_NORMAL_ONLY);
        if (ret != APP_ERR_OK) {
            aclDataBuffer* outputData = aclCreateDataBuffer(outputBuffer, bufferSize);
            if (outputData == nullptr) {
                LogError << "Create data buffer failed!";
                aclrtFree(outputBuffer);
                return APP_ERR_COMM_INVALID_POINTER;
            }
            ret = aclmdlAddDatasetBuffer(outputDataSet_.get(), outputData);
            if (ret != APP_ERR_OK) {
                LogError << "Add dataset buffer failed!";
            }
        }
    }
    return APP_ERR_OK;
}
8.9.4.2.3 Initialization Module Functions

Initialization Module Function InitModule

The following is an example of the code for initializing the module in the AclProcess.cpp file:

```cpp
#include "acl/acl.h"
//......
// Initialize the module function.
APP_ERROR AclProcess::InitModule()
{
    //Create a Dvpp jpegD decoding object by calling the library file class DvppJpegdProcess.
    if (dvppJpegD_ == nullptr) {
        dvppJpegD_ = std::make_shared<DvppJpegdProcess>(stream_);
    }
    // Create a Dvpp Resize object by calling the library file class DvppResizeProcess.
    if (dvppResize_ == nullptr) {
        dvppResize_ = std::make_shared<DvppResizeProcess>(stream_);
    }
    // Create a model inference object by calling the library file class ModelProcess.
    if (modelProcess_ == nullptr) {
        modelProcess_ = std::make_shared<ModelProcess>(deviceId_, modelInfo_.modelName);
    }
    // Call the InitResource function to initialize the model inference module. For details, see the InitResource function in Model Inference Library File ModelProcess.cpp.
    APP_ERROR ret = modelProcess_->InitResource();
    if (ret != APP_ERR_OK) {
        LogError << "ModelProcess_ Init failed!";
        return ret;
    }
    //Create the Cast operator object by calling the library file class SingleOpProcess.
    if (castOp_ == nullptr) {
        castOp_ = std::make_shared<SingleOpProcess>(stream_);
    }
    //Create the argMax operator object by calling the library file class SingleOpProcess.
    if (argMaxOp_ == nullptr) {
        argMaxOp_ = std::make_shared<SingleOpProcess>(stream_);
    }
    //Download image type lables.
    ret = LoadLabels(LABEL_PATH);
}
//......
```

8.9.4.2.4 Single-operator Information Initialization Functions

Single-operator Information Initialization Function InitOpCastResource

The following is a code sample of initializing the single-operator information using the AclProcess.cpp file:

```cpp
#include "acl/acl.h"
//......
// Initialize the Cast operator information.
APP_ERROR AclProcess::InitOpCastResource()
{
    //Call the SetTypeName function to set the cast operator name. For details, see the SetTypeName function
```
  std::string opTypeName("Cast");
  castOp_->SetTypeName(opTypeName);
  int inputTensorNum = 1;
  std::shared_ptr<aclopAttr> opAttr(aclopCreateAttr(), aclopDestroyAttr);
  OpAttr attr;
  attr.name = std::string("dst_type");
  attr.type = INT;
  attr.numInt = 1;
  //Call the SetOpAttr function to set the attributes of the cast operator. For details, see the SetOpAttr function in Single-operator Information Initialization Library File SingleOpProcess.cpp.
  APP_ERROR ret = castOp_->SetOpAttr(opAttr, attr);
  if (ret != APP_ERR_OK) {
    LogError << "OpProcess set op attribute failed";
    return ret;
  }
  //Call the SetInputTensor function to set the input tensor information of the Cast operator. For details, see the SetInputTensor function in Single-operator Information Initialization Library File SingleOpProcess.cpp.
  std::vector<Tensor> tensors = { {ACL_FLOAT, 1, {CLASS_TYPE_NUM}, ACL_FORMAT_ND},};
  castOp_->SetInputTensorNum(inputTensorNum);
  ret = castOp_->SetInputTensor(tensors);
  if (ret != APP_ERR_OK) {
    LogError << "Cast Op SetInput failed!";
    return ret;
  }
  int outputTensorNum = 1;
  //Call the SetOutputTensor function to set the output tensor information of the Cast operator. For details, see the SetOutputTensor function in Single-operator Information Initialization Library File SingleOpProcess.cpp.
  std::vector<Tensor> outTensors = { {ACL_FLOAT16, 1, {CLASS_TYPE_NUM}, ACL_FORMAT_ND},};
  castOp_->SetOutputTensorNum(outputTensorNum);
  ret = castOp_->SetOutputTensor(outTensors);
  if (ret != APP_ERR_OK) {
    LogError << "Cast Op SetOutput failed!";
    return ret;
  }
  return APP_ERR_OK;
}

Single-operator Information Initialization Function InitOpArgMaxResource

The following is a code sample of initializing the single-operator information using the AclProcess.cpp file:

```c
#include "acl/acl.h"
//......
// Initialize the ArgMax operator information.
APP_ERROR AclProcess::InitOpArgMaxResource() {
  int inputTensorNum = 1;
  std::string opTypeName = "ArgMaxD";
  //Call the SetTypeName function to set the name of the ArgMax operator. For details, see the SetTypeName function in Single-operator Information Initialization Library File SingleOpProcess.cpp.
  argMaxOp_->SetTypeName(opTypeName); // Set type name of operator
  std::shared_ptr<aclopAttr> opAttr(aclopCreateAttr(), aclopDestroyAttr); // Create attribute for operator
  OpAttr attr;
  attr.name = std::string("dimension");
  attr.type = INT;
  attr.numInt = 0;
  APP_ERROR ret = argMaxOp_->SetOpAttr(opAttr, attr);
  if (ret != APP_ERR_OK) {
    LogError << "OpProcess set op attribute failed";
    return ret;
  }
  //Call the SetInputTensor function to set the input tensor information of the ArgMax operator. For details,
```

```cpp
std::vector<Tensor> tensors = {{ACL_FLOAT16, 1, {1000}, ACL_FORMAT_ND}};
argMaxOp_->SetInputTensorNum(inputTensorNum);
ret = argMaxOp_->SetInputTensor(tensors);
if (ret != APP_ERR_OK) {
    LogError << "ArgMax Op SetInput failed!";
    return ret;
}
int argMaxOutputTensorNum = 1;
// Call the SetOutputTensor function to set the output tensor information of the ArgMax operator. For details, see the SetOutputTensor function in Single-operator Information Initialization Library File SingleOpProcess.cpp.
std::vector<Tensor> outTensors = {{ACL_INT32, 1, {1}, ACL_FORMAT_ND}};
argMaxOp_->SetOutputTensorNum(argMaxOutputTensorNum);
ret = argMaxOp_->SetOutputTensor(outTensors);
if (ret != APP_ERR_OK) {
    LogError << "ArgMax Op SetOutput failed!";
    return ret;
}
return APP_ERR_OK;
```

**Single-operator Information Initialization Library File SingleOpProcess.cpp**

The following is a code sample of initializing the single-operator information in the SingleOpProcess.cpp file:

```cpp
#include "acl/acl.h"
//......
// Call the SetInputTensor function to set the operator tensor input.
APP_ERROR SingleOpProcess::SetInputTensor(const std::vector<Tensor> &tensors) {
    inputTensorDesc_.clear();
    for (int i = 0; i < inputTensorNum_; i++) {
        // Create the input data of the aclTensorDesc type.
        std::shared_ptr<aclTensorDesc> tensorDesc(aclCreateTensorDesc(tensors[i].dataType, tensors[i].numDim, tensors[i].dims.data(), tensors[i].format), aclDestroyTensorDesc);
        if (tensorDesc == nullptr) {
            LogError << "aclCreateTensorDesc failed";
            return APP_ERR_COMM_INVALID_POINTER;
        }
        inputTensorDesc_.push_back(tensorDesc);
    }
    return APP_ERR_OK;
}
//......
// Call the SetInputDataBuffer function to set the operator tensor input data memory.
APP_ERROR SingleOpProcess::SetInputDataBuffer(const std::vector<std::shared_ptr<void>> &inputDataBuf, const std::vector<size_t> &inputBufSize) {
    inputDataBuf_.clear();
    inputData_ = inputDataBuf;
    for (int i = 0; i < inputTensorNum_; i++) {
        // Create the input data of the aclDataBuffer type.
        std::shared_ptr<aclDataBuffer> dataBuffer(aclCreateDataBuffer(inputDataBuf[i].get(), inputBufSize[i]), aclDestroyDataBuffer);
        if (dataBuffer == nullptr) {
            LogError << "aclCreateDataBuffer failed";
            return APP_ERR_COMM_INVALID_POINTER;
        }
        inputDataBuf_.push_back(dataBuffer);
    }
    return APP_ERR_OK;
}
//......
// Call the SetOutputTensor function to set the operator tensor output.
APP_ERROR SingleOpProcess::SetOutputTensor(const std::vector<Tensor> &tensors) {
    //......
```
outputTensorDesc_.clear();
outputDataBuf_.clear();
outputData_.clear();
outputDataSize_.clear();
APP_ERROR ret = APP_ERR_OK;
for (int i = 0; i < outputTensorNum_; i++) {
    // Create the output data of the aclTensorDesc type.
    std::shared_ptr<aclTensorDesc> tensorDesc(aclCreateTensorDesc(tensors[i].dataType,
tensors[i].numDim,
tensors[i].dims.data(), tensors[i].format), aclDestroyTensorDesc);
    if (tensorDesc == nullptr) {
        LogError << "aclCreateDataBuffer failed";
        return APP_ERR_COMM_INVALID_POINTER;
    }
    outputTensorDesc_.push_back(tensorDesc);
    void *outDevBuf = nullptr;
    size_t size = aclGetTensorDescSize(tensorDesc.get());
    // Allocate memory on the device.
    ret = aclrtMalloc(&outDevBuf, size, ACL_MEM_MALLOC_NORMAL_ONLY);
    if (ret != APP_ERR_OK) {
        LogError << "aclrtMalloc out result failed"
        return ret;
    }
    // Creates data of the aclDataBuffer type.
    std::shared_ptr<aclDataBuffer> dataBuf(aclCreateDataBuffer(outDevBuf, size), aclDestroyDataBuffer);
    outputDataBuf_.push_back(dataBuf);
    std::shared_ptr<void> data(outDevBuf, aclrtFree);
    outputData_.push_back(data);
    outputDataSize_.push_back(size);
}
return APP_ERR_OK;

// Call the SetOpAttr function to set the operator attribute.
APP_ERROR SingleOpProcess::SetOpAttr(const std::shared_ptr<aclopAttr> &attr, const OpAttr &attrDesc) {
    attr_ = attr;
    // Set the operator attribute by using the attrDesc.type switch.
    switch (attrDesc.type) {
    case BOOL:
        aclSetAttrBool(attr.get(), attrDesc.name.c_str(), attrDesc.numBool);
        break;
    case INT:
        aclSetAttrInt(attr.get(), attrDesc.name.c_str(), attrDesc.numInt);
        break;
    case FLOAT:
        aclSetAttrFloat(attr.get(), attrDesc.name.c_str(), attrDesc.numFloat);
        break;
    case STRING:
        aclSetAttrString(attr.get(), attrDesc.name.c_str(), attrDesc.numString.c_str());
        break;
    case LIST_BOOL: {
        const char **valuesString = new const char *[attrDesc.num];
        for (int i = 0; i < attrDesc.num; i++) {
            valuesString[i] = attrDesc.valuesString[i].c_str();
        }
        aclSetAttrListString(attr.get(), attrDesc.name.c_str(), attrDesc.num, valuesString);
        delete[] valuesString;
        break;
    }
8.9.4.3 Image Decoding and Resizing

After data is preprocessed (decoded and resized), an image in the encoding format of the source image or YUV420 SP format is output as the input for model inference. For details about data preprocessing principles, see 5.4.3 Pre-Processing Data (Image Decoding and Resizing). The ascendbase common function library is a common class and function that have been encapsulated for image decoding and resizing. You only need to call it during image decoding and resizing. The image decoding and resizing code of the InferClassification sample is implemented by the AclProcess.cpp file. Image Decoding and Resizing Function Preprocess describes the image decoding and resizing functions in the AclProcess.cpp file.

Image Decoding and Resizing Function Preprocess

The following is a code example for AclProcess.cpp to decode and resize images:

```c++
#include "acl/acl.h"

//......
// Create the image decoding and resizing function Preprocess.
APP_ERROR AclProcess::Preprocess(std::string imageFile)
{
    // Call the ProcessFile function to decode the input image in JPEG format. For details, see the ProcessFile function in Code of the DvppjpegProcess.cpp common library file.
    APP_ERROR ret = dvppjpegD_->ProcessFile(imageFile, PIXEL_FORMAT_YUV_SEMIPLANAR_420, false);
    if (ret != APP_ERR_OK) {
        LogError << "ProcessDecode failed!";
        return ret;
    }
    // Obtain the decoded image information (such as the image height and width).
    std::shared_ptr<DvppDataInfoT> decodeOutData = dvppjpegD_->GetOutput();
    // Call the ProcessDevData function to resize the decoded image. For details, see the ProcessDevData function in Code of the DvppResizeProcess.cpp library file.
    ret = dvppResize_->_ProcessDevData(decodeOutData, modelInfo_.modelWidth, modelInfo_.modelHeight, PIXEL_FORMAT_YUV_SEMIPLANAR_420, true);
    if (ret != APP_ERR_OK) {
        LogError << "ProcessResize failed!";
        return ret;
    }
    return APP_ERR_OK;
}
//......
Code of the DvppJpegdProcess.cpp common library file

The following is a code example of the DvppJpegdProcess.cpp common library file:

```c
#include "acl/acl.h"

// Create the SetOutputDesc function for decoding output images.
APP_ERROR DvppJpegdProcess::SetOutputDesc(const uint32_t outputSize)
{
    // Obtain the input data required by the image decoding model from the declaration alias DvppDataInfoT defined in the Dvpp.
    std::shared_ptr<DvppDataInfoT> input = GetInput();
    uint32_t_t inputWidth = input->width;
    uint32_t_t inputHeight = input->height;
    // Construct the image decoding output data information based on the input data information of image decoding.
    std::shared_ptr<DvppDataInfoT> output = std::make_shared<DvppDataInfoT>();
    output->width = inputWidth;
    output->height = inputHeight;
    // Align the width and height of the input image by 16 x 2 upwards.
    output->heightStride = ALIGN_UP(inputHeight, JPEGD_STRIDE_HEIGHT);
    output->widthStride = ALIGN_UP(inputWidth, JPEGD_STRIDE_WIDTH);
    output->lenOfByte = outputSize;
    output->format = input->format;
    void* decodeOutDevBuffer = nullptr;
    // 1. Apply for the device memory to store the decoded data.
    APP_ERROR ret = acldvppMalloc(&decodeOutDevBuffer, output->lenOfByte);
    if (ret != APP_ERR_OK) {
        LogError << "aclrtdvppMalloc jpegOutBufferDev failed, ret = " << ret;
        return ret;
    }
    // 1.1 Construct a smart pointer of the output data.
    output->data.reset(decodeOutDevBuffer, acldvppFree);
    SetOutput(output);
    // 1.2 Construct the image description of the decoded output data.
    acldvppPicDesc* decodeOutputDesc = acldvppCreatePicDesc();
    if (decodeOutputDesc == nullptr) {
        LogError << "acldvppCreatePicDesc decodeOutputDesc failed!";
        return APP_ERR_COMM_INVALID_POINTER;
    }
    decodeOutputDesc_.reset(decodeOutputDesc, DestroyPicDesc);
    // Set the attributes of the description of the decoded output picture.
    acldvppSetPicDescData(decodeOutputDesc_.get(), decodeOutDevBuffer);
    acldvppSetPicDescFormat(decodeOutputDesc_.get(), output->format);
    acldvppSetPicDescWidth(decodeOutputDesc_.get(), inputWidth);
    acldvppSetPicDescHeight(decodeOutputDesc_.get(), inputHeight);
    acldvppSetPicDescWidthStride(decodeOutputDesc_.get(), output->widthStride);
    acldvppSetPicDescHeightStride(decodeOutputDesc_.get(), output->heightStride);
    acldvppSetPicDescSize(decodeOutputDesc_.get(), outputSize);
    return APP_ERR_OK;
}

// Create the TransferImageH2D function for copying image data from the host to the device.
APP_ERROR DvppJpegdProcess::TransferImageH2D(const FileInfo& imageInfo, const uint32_t imageWidth, const uint32_t imageHeight, const std::shared_ptr<DvppDataInfoT>& jpegInput)
{
    // Check the memory size.
    if (imageInfo.lenOfByte <= 0) {
        LogError << "The inputHostBufferSize should not empty!";
        return APP_ERR_COMM_INVALID_PARAM;
    }
    uint32_t_t inBufferSize = imageInfo.lenOfByte + JPEG_OFFSET;
    void* inDevBuff = nullptr;
    APP_ERROR ret = acldvppMalloc(&inDevBuff, inBufferSize);
    if (ret != APP_ERR_OK) {
        LogError << "DvppMalloc device buffer failed, size is " << inBufferSize;
        return ret;
    }
    std::shared_ptr<void> inputDeviceBuff;
    inputDeviceBuff.reset(inDevBuff, acldvppFree);
    aclrtStream stream = GetStream();
```
// Copy image data from the host to the device.
ret = aclrtMemcpyAsync(inputDeviceBuff.get(), inBufferSize, imageInfo.data.get(),
    imageInfo.lenOfByte, ACL_MEMCPY_HOST_TO_DEVICE, stream);
if (ret != APP_ERR_OK) {
    LogError << "memcpy failed device buffer size is " << inBufferSize <<
        " input host buffer size is " << imageInfo.lenOfByte;
    return ret;
}
jpegInput->data = inputDeviceBuff;
jpegInput->lenOfByte = inBufferSize;
jpegInput->width = imageWidth;
jpegInput->height = imageHeight;
return APP_ERR_OK;
}
// Create the ProcessFile function to read image data from the image file.
APP_ERROR DvppJpegdProcess::ProcessFile(const std::string& imageFile,
    const acldvppPixelFormat format, const bool withSynchronize)
{
    FileInfo imageInfo;
    APP_ERROR ret = ReadFile(imageFile, imageInfo);
    if (ret != APP_ERR_OK) {
        LogError << "DvppJpegd read image file failed";
        return ret;
    }
    return ProcessHostData(imageInfo, format, withSynchronize);
}
// Create the ProcessHostData function for reading image information on the host.
APP_ERROR DvppJpegdProcess::ProcessHostData(const FileInfo& imageInfo,
    const acldvppPixelFormat format, const bool withSynchronize)
{
    uint32_t width;
    uint32_t height;
    int32_t components;
    // Reads the width and height of the JPEG image from the memory on the host that stores the JPEG
    image data.
    APP_ERROR ret = acldvppJpegGetImageInfo(imageInfo.data.get(), imageInfo.lenOfByte, &width, &height,
        &components);
    if (ret != APP_ERR_OK) {
        LogError << "Get image width and height of JPEG failed";
        return ret;
    }
    // Predicts the output memory size required for the decoded JPEG image based on the memory used for
    storing JPEG image data on the host.
    uint32_t outputSize;
    ret = acldvppJpegPredictDecSize(imageInfo.data.get(), imageInfo.lenOfByte, format, &outputSize);
    if (ret != APP_ERR_OK) {
        LogError << "Predict decode size of JPEG image failed";
        return ret;
    }
    std::shared_ptr<DvppDataInfoT> jpegInput = std::make_shared<DvppDataInfoT>();
    jpegInput->format = format;
    ret = TransferImageH2D(imageInfo, width, height, jpegInput);
    if (ret != APP_ERR_OK) {
        LogError << "Transfer buffer from host to device failed";
        return ret;
    }
    return ProcessDevData(jpegInput, outputSize, format, withSynchronize);
}
// Create the ProcessDevData function to perform asynchronous decoding
APP_ERROR DvppJpegdProcess::ProcessDevData(const std::shared_ptr<DvppDataInfoT> jpegInput, const
    uint32_t outputSize,
    const acldvppPixelFormat format, const bool withSynchronize)
{
    if (jpegInput->width > MAX_JPEGD_WIDTH || jpegInput->height > MAX_JPEGD_HEIGHT) {
        LogError << "Image width or height is larger than 8192";
        return APP_ERR_COMM_INVALID_PARAM;
    }
    if (jpegInput->width < MIN_JPEGD_WIDTH || jpegInput->height < MIN_JPEGD_HEIGHT) {
        LogError << "Image width or height is smaller than 32";
        return APP_ERR_COMM_INVALID_PARAM;
    }
Code of the DvppResizeProcess.cpp library file

The following is a code example of the DvppResizeProcess.cpp common library file:

```cpp
#include "acl/acl.h"

// Create the SetResizeParam function for setting image resizing parameters.
APP_ERROR DvppResizeProcess::SetResizeParam(uint32_t resizeWidth, uint32_t resizeHeight, acldvppPixelFormat format)
{
    // Check the image resizing parameters.
    if ((resizeWidth < MIN_RESIZE_WIDTH) || (resizeHeight < MIN_RESIZE_HEIGHT))
    {
        LogError << "resize width and height invalid, should be in \[32, 4096\], which are "
                      << resizeWidth << " and " << resizeHeight;
        return APP_ERR_COMM_INVALID_PARAM;
    }
    if ((resizeWidth > MAX_RESIZE_WIDTH) || (resizeHeight > MAX_RESIZE_HEIGHT))
    {
        LogError << "resize width and height invalid, should be in \[32, 4096\], which are "
                      << resizeWidth << " and " << resizeHeight;
        return APP_ERR_COMM_INVALID_PARAM;
    }
    // Set the resizing configuration.
    resizeConfig_.reset(acldvppCreateResizeConfig(), DestroyResizeConfig);
    if (resizeConfig_.get() == nullptr) {
        LogError << "acldvppCreateResizeConfig failed!";
        return APP_ERR_COMM_INVALID_POINTER;
    }
    // Align the resized image upwards.
    uint32_t resizeWidthStride = ALIGN_UP(resizeWidth, VPC_STRIDE_WIDTH);
    uint32_t resizeHeightStride = ALIGN_UP(resizeHeight, VPC_STRIDE_HEIGHT);
    uint32_t resizeOutBufferSize = resizeWidthStride * resizeHeightStride * YUV_BYTES_NU / YUV_BYTES_DE;
    void* resizeOutBufferDev = nullptr;
    // Apply for the resizing output memory resizeOutBufferDev.
    APP_ERROR ret = acldvppMalloc(&resizeOutBufferDev, resizeOutBufferSize);
    if (ret != APP_ERR_OK) {
        LogError << "aclrtMalloc resizeOutBuffer, failed, ret = " << ret;
        return ret;
    }
// Create the description of the output image and set the attribute values.
std::shared_ptr<DvppDataInfoT> output = std::make_shared<DvppDataInfoT>();
output->width = resizeWidth;
output->height = resizeHeight;
output->widthStride = resizeWidthStride;
output->heightStride = resizeHeightStride;
output->data.reset(resizeOutBufferDev, acldvppFree);
output->lenOfByte = resizeOutBufferSize;
output->format = format;
SetOutput(output);
return APP_ERR_OK;
}

// Create the UpdateInputDesc function to initialize the input description of image decoding.
APP_ERROR DvppResizeProcess::UpdateInputDesc()
{
    std::shared_ptr<DvppDataInfoT> input = GetInput();
    uint32_t inputWidthStride;
    if (input->format >= PIXEL_FORMAT_YUV_400 && input->format <=
        PIXEL_FORMAT_YUV_SEMIPLANAR_444) {
        inputWidthStride = ALIGN_UP(input->widthStride, VPC_STRIDE_WIDTH);
        } else if (input->format >= PIXEL_FORMAT_YUV_PACKED_422 && input->format <=
        PIXEL_FORMAT_YUVY_PACKED_422) {
        inputWidthStride = ALIGN_UP(input->widthStride, VPC_STRIDE_WIDTH) * YUV422_WIDTH_NU;
        } else if (input->format >= PIXEL_FORMAT_YUV_PACKED_444 && input->format <=
        PIXEL_FORMAT_BGR_888) {
        inputWidthStride = ALIGN_UP(input->widthStride, VPC_STRIDE_WIDTH) * YUV444_RGB_WIDTH_NU;
        } else if (input->format >= PIXEL_FORMAT_ARGB_8888 && input->format <=
        PIXEL_FORMAT_BGRA_8888) {
        inputWidthStride = ALIGN_UP(input->widthStride, VPC_STRIDE_WIDTH) * XRGB_WIDTH_NU;
        } else {
            LogError << "Input format is invalid! Please check it!";
            return APP_ERR_COMM_INVALID_PARAM;
        }
    uint32_t inputHeightStride = ALIGN_UP(input->heightStride, VPC_STRIDE_HEIGHT);

    // Check whether the width of the image after zooming is correct.
    if (input->widthStride > MAX_RESIZE_WIDTH || input->widthStride < MIN_RESIZE_WIDTH) {
        LogError << "Input width stride is invalid! range = [32, 4096]!";
        return APP_ERR_COMM_INVALID_PARAM;
    }

    // Check whether the height of the image after zooming is correct.
    if (input->heightStride > MAX_RESIZE_HEIGHT || input->heightStride < MIN_RESIZE_HEIGHT) {
        LogError << "Input width stride is invalid! range = [32, 4096]!";
        return APP_ERR_COMM_INVALID_PARAM;
    }

    acldvppPicDesc* resizeInputDesc = acldvppCreatePicDesc();
    if (resizeInputDesc == nullptr) {
    LogError << "InitResizeInputDesc failed";
    return APP_ERR_COMM_INVALID_POINTER;
    }

    // Create the description of the input image and set the attribute values.
    resizeInputDesc->reset(resizeInputDesc, DestroyPicDesc);
    acldvppSetPicDescData(resizeInputDesc_get(), input->data.get());
    acldvppSetPicDescFormat(resizeInputDesc_get(), input->format);
    acldvppSetPicDescWidth(resizeInputDesc_get(), input->width);
    acldvppSetPicDescHeight(resizeInputDesc_get(), input->height);
    acldvppSetPicDescWidthStride(resizeInputDesc_get(), inputWidthStride);
    acldvppSetPicDescHeightStride(resizeInputDesc_get(), inputHeightStride);
    acldvppSetPicDescSize(resizeInputDesc_get(), input->lenOfByte);
    return APP_ERR_OK;
}

// Create the UpdateOutputDesc function to initialize the image decoding output description.
APP_ERROR DvppResizeProcess::UpdateOutputDesc()
{
    //Create the image pixel description of the DVPP.
    acldvppPicDesc* resizeOutputDesc = acldvppCreatePicDesc();
    if (resizeOutputDesc == nullptr) {
            LogError << "acldvppCreatePicDesc failed";
        return APP_ERR_COMM_INVALID_POINTER;
    }
// Create the description of the output image and set the attribute values.
.resizeOutputDesc_.reset(resizeOutputDesc, DestroyPicDesc);
std::shared_ptr<DvppDataInfoT> output = GetOutput();
acldvppSetPicDescData(resizeOutputDesc_.get(), output->data.get());
acldvppSetPicDescFormat(resizeOutputDesc_.get(), output->format);
acldvppSetPicDescWidth(resizeOutputDesc_.get(), output->width);
acldvppSetPicDescHeight(resizeOutputDesc_.get(), output->height);
acldvppSetPicDescWidthStride(resizeOutputDesc_.get(), output->widthStride);
acldvppSetPicDescHeightStride(resizeOutputDesc_.get(), output->heightStride);
acldvppSetPicDescSize(resizeOutputDesc_.get(), output->lenOfByte);
return APP_ERR_OK;
}

// Check the resizing parameters.
APP_ERROR DvppResizeProcess::CheckParams(std::shared_ptr<DvppDataInfoT> inputData,
uint32_t resizeWidth, uint32_t resizeHeight, acldvppPixelFormat format)
{
if (format != PIXEL_FORMAT_YUV_SEMIPLANAR_420 && format !=
PIXEL_FORMAT_YVU_SEMIPLANAR_420) {
    LogError << "Output format of dvpp resize is invalide, just support NV12 or NV21!";
    return APP_ERR_COMM_INVALID_PARAM;
}

if (((float)resizeHeight / inputData->height) < MIN_RESIZE_SCALE ||
((float)resizeHeight / inputData->height) > MAX_RESIZE_SCALE) {
    LogError << "Resize scale should be in range [1/16, 32], which is " <<
    resizeHeight / inputData->height << "; !";
    return APP_ERR_COMM_INVALID_PARAM;
}

if (((float)resizeWidth / inputData->width) < MIN_RESIZE_SCALE ||
((float)resizeWidth / inputData->width) > MAX_RESIZE_SCALE) {
    LogError << "Resize scale should be in range [1/16, 32], which is " <<
    resizeWidth / inputData->width << " !";
    return APP_ERR_COMM_INVALID_PARAM;
}
return APP_ERR_OK;
}

// Create the ProcessDevData function to execute the asynchronous resizing interface.
APP_ERROR DvppResizeProcess::ProcessDevData(std::shared_ptr<DvppDataInfoT> inputData, uint32_t
resizeWidth,
uint32_t resizeHeight, acldvppPixelFormat format, bool withSynchronize)
{
    APP_ERROR ret = CheckParams(inputData, resizeWidth, resizeHeight, format);
    if (ret != APP_ERR_OK) {
        return ret;
    }

    // Set the input data of the resizing model.
    SetInput(inputData);
    ret = SetResizeParam(resizeWidth, resizeHeight, format);
    if (ret != APP_ERR_OK) {
        LogError << "SetResizeParam failed!";
        return ret;
    }

    // Initialize the description of the input data for picture resizing.
    ret = UpdateInputDesc();
    if (ret != APP_ERR_OK) {
        LogError << "InitInputDesc of DvppResize failed!";
        return ret;
    }

    // Initialize the description of the output data of image resizing.
    ret = UpdateOutputDesc();
    if (ret != APP_ERR_OK) {
        LogError << "InitOutputDesc of DvppResize failed!";
        return ret;
    }

    // Obtain the stream of the resizing model.
    aclrtStream stream = GetStream();
    // Obtain the description of the resizing channel.
    std::shared_ptr<acldvppChannelDesc> dvppChannelDesc = GetChannelDesc();
    // Resize the input image.
    ret = acldvppVpcResizeAsync(dvppChannelDesc.get(), resizeInputDesc_.get());
8.9.4.4 Image Inference and Classification

The decoded and resized image data can be added to the input model for inference. For details about the model inference principles, see 5.4.4 Performing Model Inference (Batch Size = 1, Fixed Shape, Static AIPP, Single Model). The ascendbase common function library contains common classes and functions that have been encapsulated by developers for image inference and classification. You only need to call them during image inference and classification. The image inference and classification code of the InferClassification sample is implemented by the AclProcess.cpp file. Image Inference Classification Function ModelInfer shows the image inference and classification functions in the AclProcess.cpp file.

Image Inference Classification Function ModelInfer

The following is a code sample of image inference and classification in the AclProcess.cpp file:

```c
#include "acl/acl.h"

//......

namespace acl { struct timeval begin, end; }

// Create the model inference function ModelInfer.
APP_ERROR AclProcess::ModelInfer()
{
  // Call the GetOutput function to obtain the resized image data. See DvppProcess.cpp Library File: GetOutput Function.
  std::shared_ptr<DvppDataInfoT> resizeOutData = dvppResize_->GetOutput();
  struct timeval begin = {0};
  struct timeval end = {0};
  gettimeofday(&begin, nullptr);
  // Specify the data flow and size of the input model.
  std::vector<std::shared_ptr<void>> inputDataBuffers({ resizeOutData->data });
  std::vector<size_t> buffersSize({ resizeOutData->lenOfByte });
  APP_ERROR ret = modelProcess_->ProcessDevBuf(inputDataBuffers, buffersSize);"
DvppProcess.cpp Library File: GetOutput Function

DvppProcess.cpp library file: GetOutput function

```cpp
#include "acl/acl.h"

// Create a GetOutput function of the DvppDataInfoT type.
// Declare the alias DvppDataInfoT. For details, see typedef struct DvppDataInfo in DvppProcess.h
std::shared_ptr<DvppDataInfoT> DvppProcess::GetOutput()
{
    return output_;
}
```


ModelProcess.cpp library file: ProcessDevBuf function

```cpp
#include "acl/acl.h"

// Create the ProcessDevBuf function to perform model synchronization inference.
APP_ERROR ModelProcess::ProcessDevBuf(const std::vector<std::shared_ptr<void>>& inputDevBufs,
                                      const std::vector<size_t>& buffersSize)
{
    // Initialize model inference resources.
    if (initFlag != true) {
        APP_ERROR ret = InitResource();
        if (ret != APP_ERR_OK) {
            LogError << "ModelProcess Init failed!";
            return ret;
        }
    }
    // Set the input memory.
    APP_ERROR ret = SetInput(inputDevBufs, buffersSize);
    if (ret != APP_ERR_OK) {
        LogError << "model create input failed!";
        return ret;
    }
    // Perform model inference.
    ret = aclmdlExecute(modelId_, inputDataSet_.get(), outputDataSet_.get());
    if (ret != APP_ERR_OK) {
        LogError << "Excute model failed, modelId_ is " << modelId_;
        return ret;
    }
    return APP_ERR_OK;
}
```

8.9.4.5 Post-classification Image Processing

After model inference is complete, post-process the model inference result. After data postprocessing is complete, release related resources. For details about the post-classification image processing, see 8.9.4.5 Post-classification Image Processing. The post-classification image processing code of the InferClassification sample is implemented by the AclProcess.cpp file. Table 8-10 shows post-classification image processing functions in the AclProcess.cpp file.
### Table 8-10 InferClassification post-classification image processing functions

<table>
<thead>
<tr>
<th>Post-classification Processing Function</th>
<th>Description</th>
<th>Sample Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>CastOpInfer</td>
<td>Converts the data type of the inference result from Float32 to Float16.</td>
<td>CastOpInfer Function for Post-classification Image Processing</td>
</tr>
<tr>
<td>ArgMaxOpInfer</td>
<td>Searches or the category ID with the maximum confidence in the inference result.</td>
<td>ArgMaxOpInfer Function for Post-classification Image Processing</td>
</tr>
<tr>
<td>PostProcess</td>
<td>Prints the class label with the highest confidence score in the terminal console.</td>
<td>PostProcess Function for Post-classification Image Processing</td>
</tr>
</tbody>
</table>

**CastOpInfer Function for Post-classification Image Processing**

The following is an example of the `AclProcess.cpp` code file for post-classification image processing:

```cpp
#include "acl/acl.h"

//......

// Create the CastOpInfer inference function CastOpInfer to convert the data type of the inference result from float32 to float16.
APP_ERROR AclProcess::CastOpInfer()
{
    // Obtain the inference output data of the ResNet50 model.
    std::vector<RawDevData> modelOutput = modelProcess_->getModelOutputData();
    if (modelOutput.size() <= 0) {
        LogError << "Get model output data failed";
        return APP_ERR_INFER_GET_OUTPUT_FAIL;
    }

    // Construct the memory and size of the input data of the Cast operator
    std::vector<std::shared_ptr<void>> inputDataBuf({modelOutput[0].data});
    std::vector<size_t> inputBufSize({modelOutput[0].lenOfByte});
    castOp_->SetInputDataBuffer(inputDataBuf, inputBufSize);

    struct timeval begin = {0};
    struct timeval end = {0};
    gettimeofday(&begin, nullptr);

    APP_ERROR ret = castOp_->RunSingleOp(true);
    gettimeofday(&end, nullptr);

    // Calculate the running time of the Cast operator.
    const double costMs = SEC2MS * (end.tv_sec - begin.tv_sec) + (end.tv_usec - begin.tv_usec) / SEC2MS;
    const double fps = 1 / costMs;
    if (ret != APP_ERR_OK) {
        LogError << "Cast Op execute failed";
        return ret;
    }
    return APP_ERR_OK;
}
//......
```

Atlas Intelligent Edge Solution
Application Software Development Guide
Issue 01 (2020-07-30)
ArgMaxOpInfer Function for Post-classification Image Processing

The following is an example of the AclProcess.cpp code file for post-classification image processing:

```cpp
#include "acl/acl.h"

// Create the ArgMaxOp inference function ArgMaxOpInfer to search for the class identifier of the maximum confidence level from the inference result.
APP_ERROR AclProcess::ArgMaxOpInfer()
{
    // Obtain the inference output data of the Cast operator.
    std::vector<std::shared_ptr<void>> castOpOutput = castOp_->GetOutputData();
    std::vector<size_t> outDataSize = castOp_->GetOutputDataSize();
    std::shared_ptr<void> castOpOutputDataPtr = castOpOutput[0];
    // Construct the memory and size of the input data of the ArgMax operator.
    std::vector<std::shared_ptr<void>> inputDataBuf({castOpOutputDataPtr});
    std::vector<size_t> inputBufSize({outDataSize});
    argMaxOp_->SetInputDataBuffer(inputDataBuf, inputBufSize);
    struct timeval begin = {0};
    struct timeval end = {0};
    gettimeofday(&begin, nullptr);
    APP_ERROR ret = argMaxOp_->RunSingleOp(true);
    gettimeofday(&end, nullptr);
    // Calculate the running time of the ArgMax operator.
    const double costMs = SEC2MS * (end.tv_sec - begin.tv_sec) + (end.tv_usec - begin.tv_usec) / SEC2MS;
    const double fps = 1 * SEC2MS / costMs;
    if (ret != APP_ERR_OK) {
        LogError << "ArgMax Op execute failed";
        return ret;
    }
    // Block application execution until all tasks in a specified stream are complete. It is a synchronous interface.
    ret = aclrtSynchronizeStream(stream_);
    if (ret != APP_ERR_OK) {
        LogError << "aclopExecute aclrtSynchronizeStream failed";
        return ret;
    }
    return APP_ERR_OK;
}
```

PostProcess Function for Post-classification Image Processing

The following is an example of the AclProcess.cpp code file for post-classification image processing:

```cpp
#include "acl/acl.h"

// Create PostProcess and print the class label with the highest confidence score in the terminal console.
APP_ERROR AclProcess::PostProcess()
{
    // Obtain the output data of the ArgMax operator.
    std::vector<std::shared_ptr<void>> argMaxOpOutput = argMaxOp_->GetOutputData();
    std::vector<size_t> outDataSize = argMaxOp_->GetOutputDataSize();
    void* argMaxOpOutputData = argMaxOpOutput[0].get();
    // Apply for the memory for storing the output data of the ArgMax operator on the host.
    void *resHostBuf = nullptr;
    APP_ERROR ret = aclrtMallocHost(&resHostBuf, outDataSize[0]);
    if (ret != APP_ERR_OK) {
        LogError << "fail to print result, malloc host failed";
        return ret;
    }
    std::shared_ptr<void> outBuf(resHostBuf, aclrtFreeHost);
    // Copy the output data of the ArgMax operator from the device to the host.
    ret = aclrtMemcpy(outBuf.get(), outDataSize[0], argMaxOpOutputData, 0);
    if (ret != APP_ERR_OK) {
        LogError << "fail to copy result, memcpy failed";
        return ret;
    }
    return APP_ERR_OK;
}
```
if (ret != APP_ERR_OK) {
    LogError << "fail to print result, memcpy device to host failed";
    return ret;
}

// Print the class label with the highest confidence score in the terminal console.
int32_t *index = static_cast<int32_t *>(resHostBuf);

// Call the WriteResult function in the AclProcess.cpp code file to write the result to a .txt file.
ret = WriteResult("result", *index);
if (ret != APP_ERR_OK) {
    LogError << "write result file failed!";
    return ret;
}

return APP_ERR_OK;

//......


#include "acl/acl.h"

//......

// Create the RunSingleOp function to execute a specified operator.
APP_ERROR SingleOpProcess::RunSingleOp(const bool &withHandle)
{
    withHandle_ = withHandle;

    // Convert inputTensorDesc to aclTensorDesc ** to execute a single operator or create a handle.
    std::vector<aclTensorDesc *> inTensorDesc;
    for (auto tensor : inputTensorDesc_) {
        inTensorDesc.push_back(tensor.get());
    }

    // Convert inputDataBuf to aclDataBuffer ** to execute a single operator.
    std::vector<aclDataBuffer *> inData;
    for (auto data : inputDataBuf_) {
        inData.push_back(data.get());
    }

    // Convert outputTensorDesc to aclTensorDesc ** to execute a single operator or create a handle.
    std::vector<aclTensorDesc *> outTensorDesc;
    for (auto tensor : outputTensorDesc_) {
        outTensorDesc.push_back(tensor.get());
    }

    // Convert outputDataBuf to aclDataBuffer ** to execute a single operator.
    std::vector<aclDataBuffer *> outData;
    for (auto data : outputDataBuf_) {
        outData.push_back(data.get());
    }

    APP_ERROR ret = APP_ERR_OK;
    if (withHandle_ == true) {
        // Create a handle.
        ret = (typeName_.c_str(), inputTensorNum_, inTensorDesc.data(), outputTensorNum_, outTensorDesc.data(), attr_.get(), &opHandle_);
        if (ret != APP_ERR_OK) {
            LogError << typeName_ << "CreateHandle failed";
            return ret;
        }

        // Call a specified operator by using a handle.
        ret = aclopExecWithHandle(opHandle_, inputTensorNum_, inputData.data(), inputTensorNum_, outData.data(), stream_);
        if (ret != APP_ERR_OK) {
            LogError << typeName_ << "ExecWithHandle failed";
            return ret;
        }
    } else {
        // Call aclopExecute to asynchronously load and execute a specified operator.
        ret = aclopExecute(typeName_.c_str(), inputTensorNum_, inTensorDesc.data(), inputData.data(), outputTensorNum_, outTensorDesc.data(), outData.data(), attr_.get(), stream_);
        if (ret != APP_ERR_OK) {
            LogError << "execute singleOp " << typeName_ << " failed";
            return ret;
        }
    }
}
8.9.4.6 Resource Release and Deinitialization

When the InferClassification program stops, the DVPP, Cast, and ArgMax operators and running management resources need to be released and deinitialized. For details about the resource release and deinitialization principles, see 8.9.4.6 Resource Release and Deinitialization.


Resource Release and Deinitialization Function: Release

The following is a Release code sample in the AclProcess.cpp code file for releasing resources and deinitializing resources:

```c++
#include "acl/acl.h"

//.....
void AclProcess::Release()
{
    APP_ERROR ret;
    // Release DVPP object resources.
    dvppJpegD_.reset();
    dvppResize_.reset();
    // Release models.
    modelProcess_.reset();
    // Release operators.
    castOp_.reset();
    argMaxOp_.reset();
    // Release DVPP object resources.
    dvppChannelDesc_.reset();
    // Release streams.
    if (stream_ != nullptr) {
        // Block application execution until all the preceding tasks in a specified stream are complete. It is a synchronous interface.
        ret = aclrtSynchronizeStream(stream_);
        if (ret != APP_ERR_OK) {
            LogError << "aclrtSynchronizeStream before destroy failed, ret = " << ret;
        }
        // Destroy a specified stream after all tasks in the stream are complete by calling aclrtSynchronizeStream.
        ret = aclrtDestroyStream(stream_);
        if (ret != APP_ERR_OK) {
            LogError << "Destroy stream failed!";
        }
        stream_ = nullptr;
    }
    //.....
```
8.9.5 Code Compilation and Running

Compilation Tool

The advantage of CMake is that it is a cross-platform compilation tool. After you write CMakeList.txt, CMake automatically generates the makefile or project file that matches the environment.

For details about the cmake parameters, visit https://cmake.org/cmake/help/latest/guide/tutorial/index.html and query the required version.

Making Preparations

You need to obtain the ResNet-50 model, convert the model into an OM file by following the instructions provided in Example (Converting a Caffe Model into an Offline Model), and upload the OM file to the InferClassification/data/models/resnet directory.

Convert the op_list.json operator file in the InferClassification/data/models/single_op directory to the OM file by following the instructions provided in Example (Converting a Single-Operator JSON File into an Offline Model), and upload the OM file to the InferClassification/data/models/single_op directory. Upload the test image and sample image to the InferClassification/data directory.

Compilation and Running

This document uses HwHiAiUser as an example.

Step 1 Build the code.

1. Log in to the development environment as the HwHiAiUser user.
2. Set environment variables. Replace xxx-linux_gccx.x.x with the actual installation path. Perform one of the following steps based on the actual situation:

   If the OS architecture of the development environment is the same as that of the operating environment, run the following commands to import environment variables:

   ```
   export ASCEND_HOME=/home/HwHiAiUser/Ascend
   export ASCEND_VERSION=/ascend-toolkit/latest
   export ARCH_PATTERN=/xxx-linux_gccx.x.x
   export LD_LIBRARY_PATH=$ASCEND_HOME/$ASCEND_VERSION/$ARCH_PATTERN/acllib/lib64:$LD_LIBRARY_PATH
   ```

   When the operating environment is deployed on Atlas 500 AI edge stations, if the development environment uses the x86 architecture and the operating environment uses the ARM architecture, the ACLlib library of the ARM architecture development kit needs to be called during application compilation because the development kits of both the x86 and ARM architectures are deployed in the development environment. Therefore, you need to import the environment variables to the path of the ACLlib library in the ARM architecture.

   ```
   export ASCEND_HOME=/home/HwHiAiUser/Ascend
   export ASCEND_VERSION=/ascend-toolkit/latest
   export ARCH_PATTERN=/arm64-linux_gcc7.3.0
   ```
3. Go to the root directory of the developed code folder and create the `CMakeList.txt` file. You can compile the file content as required. For details about the file compilation method and example, see [CMakelist Compilation Sample](#).

4. Go to the **InferClassification** directory and create a directory for storing build outputs. For example, the directory created in this sample is `build/intermediates/host`.

   ```
   mkdir -p build/intermediates/host
   ```

5. Go to the `build/intermediates/host` directory and run the `cmake` command.

   ```
   ../../../ indicates the directory where the CMakeLists.txt file is located. Replace it with the actual path.
   ```

   ```
   cd build/intermediates/host
   ```

   **Note:** If the OS architecture of the development environment is the same as that of the operating environment, run the following command to perform compilation:

   ```
   cmake ../../../src -DCMAKE_CXX_COMPILER=g++ -DCMAKE_SKIP_RPATH=TRUE
   ```

   If the operating environment is deployed on an Atlas 500 AI edge station, the development environment uses the x86 architecture, and the operating environment uses the ARM architecture, run the following command to perform cross compilation:

   ```
   cmake ../../../src -DCMAKE_CXX_COMPILER=aarch64-linux-gnu-g++ -DCMAKE_SKIP_RPATH=TRUE
   ```

6. Run the `make` command.

   The generated executable file `main` is stored in the `dist` directory of the root directory.

7. Switch to the **InferClassification** directory and run the following commands to copy the files in the `data` directory to the `dist` directory:

   ```
   cd dist
   mkdir data
   cp -r ../data/* ./data
   ```

---

**Step 2** Run the code:

1. Log in to any directory on the host as the `HwHiAiUser` user.

2. Set the environment variable.

   ```
   export LD_LIBRARY_PATH=/home/HwHiAiUser/Ascend/nnrt/latest/xxx-linux_gccx.x.x/acllib/lib64:/usr/local/Ascend/add-ons
   ```

3. Upload the compiled code folder to the current directory.

4. Go to the directory where the executable file `main` is located, that is, the `dist` directory in the root directory of the code folder.

5. Run the `.main -i data/test.jpg` command to run the executable file (`test.jpg` is the test image) to obtain the sample running result.

   ```
   ```
CMakelist Compilation Sample

Understanding the compilation commands in the CMakeList.txt file helps you understand the demo compilation process. Table 8-11 describes the main compilation statements used in the CMakeList.txt file of the InferClassification sample.

<table>
<thead>
<tr>
<th>Compilation Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmake_minimum_required</td>
<td>Specifies the earliest version of CMake required for compilation.</td>
</tr>
<tr>
<td>project</td>
<td>Specifies the project name.</td>
</tr>
<tr>
<td>add_definitions</td>
<td>Defines the macros required during compilation.</td>
</tr>
<tr>
<td>set</td>
<td>Assign values to the variables used during compilation.</td>
</tr>
<tr>
<td></td>
<td>Example: set(CMAKE_SKIP_BUILD_RPATH True)</td>
</tr>
<tr>
<td></td>
<td>The statement sets the CMAKE_SKIP_BUILD_RPATH variable to True.</td>
</tr>
<tr>
<td>get_filename_component</td>
<td>Obtains the file path.</td>
</tr>
<tr>
<td></td>
<td>Example: get_filename_component(ASCEND_BASE_ABS_DIR ${FILE} ABSOLUTE)</td>
</tr>
<tr>
<td></td>
<td>The statement assigns the absolute path of the file to ASCEND_BASE_ABS_DIR.</td>
</tr>
<tr>
<td>file</td>
<td>Obtains files recursively.</td>
</tr>
<tr>
<td></td>
<td>Example: file(GLOBRECURSE src_file RELATIVE $ (PROJECT_SOURCE_DIR) src/*.cpp)</td>
</tr>
<tr>
<td></td>
<td>The statement indicates that files in the src directory and its subdirectories are obtained recursively and assigned to src_file.</td>
</tr>
<tr>
<td>add_compile_options</td>
<td>Adds compilation options.</td>
</tr>
</tbody>
</table>
### Compilation Statement Description

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>include_directories</td>
<td>Includes the header files used by the project during compilation.</td>
</tr>
<tr>
<td></td>
<td>Example: include_directories (/usr/local/Ascend/driver/include/dvpp/)</td>
</tr>
<tr>
<td></td>
<td>The statement indicates that all header files in the /usr/local/Ascend/driver/include/dvpp/ directory are included.</td>
</tr>
<tr>
<td>aux_source_directory</td>
<td>Searches for all source files in a path.</td>
</tr>
<tr>
<td></td>
<td>Example: aux_source_directory(${PROJECT_SRC_ROOT}/CommandLine SCALE_FILES)</td>
</tr>
<tr>
<td></td>
<td>This statement searches for all source files in the CommandLine directory and assigns the files to SRC_FILES.</td>
</tr>
<tr>
<td>link_directories</td>
<td>Adds the library dependency directories, including the static library and dynamic library.</td>
</tr>
<tr>
<td></td>
<td>Example: link_directories(${DRV_LIB_DIR} ${ACL_LIB_DIR})</td>
</tr>
<tr>
<td></td>
<td>The statement indicates that the library files in the DRV_LIB_DIR and ACL_LIB_DIR directories are linked during compilation. Note that DRV_LIB_DIR and ACL_LIB_DIR are absolute paths.</td>
</tr>
<tr>
<td>add_executable</td>
<td>Generates executable programs.</td>
</tr>
<tr>
<td></td>
<td>Example: add_executable(main ${SRC_FILES} ${ASCEND_BASE_SRC_FILES} ${PROJECT_SRC_ROOT}/main.cpp)</td>
</tr>
<tr>
<td></td>
<td>The statement compiles the contents of SRC_FILES, ASCEND_BASE_SRC_FILES, and main.cpp into the executable file main.</td>
</tr>
<tr>
<td>target_link_libraries</td>
<td>Sets the name of the library file to be linked.</td>
</tr>
<tr>
<td></td>
<td>Example: target_link_libraries(main ascendcl acl_dvpp c_sec pthread)</td>
</tr>
<tr>
<td></td>
<td>The statement indicates that the names of the library files to be linked by main are ascendcl, acl_dvpp, c_sec, and pthread.</td>
</tr>
</tbody>
</table>

The content of the CMakeList.txt file of the InferClassification sample is as follows:

```cmake
# The earliest version of CMake on which compilation depends is 3.5.1.
cmake_minimum_required(VERSION 3.5.1)
# Add the macro definition DENABLE_ACL_DVPP_INTERFACE for using the DVPP APIs.
add_definitions(-DENABLE_DVPP_INTERFACE)
# Set the project name to InferClassification.
project(InferClassification)
# If the ASCEND_HOME environment variable is not set, the system generates a fatal log and stops the
```

---

The content of the CMakeList.txt file of the InferClassification sample is as follows:

```cmake
# The earliest version of CMake on which compilation depends is 3.5.1.
cmake_minimum_required(VERSION 3.5.1)
# Add the macro definition DENABLE_ACL_DVPP_INTERFACE for using the DVPP APIs.
add_definitions(-DENABLE_DVPP_INTERFACE)
# Set the project name to InferClassification.
project(InferClassification)
# If the ASCEND_HOME environment variable is not set, the system generates a fatal log and stops the
```
compilation.
if(NOT DEFINED ENV{ASCEND_HOME})
    message(FATAL_ERROR "please define environment variable:ASCEND_HOME")
endif()
# If the DRIVER_HOME environment variable is not set, set it to ASCEND_HOME. Otherwise, set it to the
# corresponding environment variable.
if(NOT DEFINED ENV{DRIVER_HOME})
    set(DRIVER_HOME $ENV{ASCEND_HOME})
else()
    set(DRIVER_HOME $ENV{DRIVER_HOME})
endif()
# If the ASCEND_VERSION environment variable is not set, the system generates a fatal log and stops the
# compilation.
if(NOT DEFINED ENV{ASCEND_VERSION})
    message(FATAL_ERROR "please define environment variable:ASCEND_VERSION")
endif()
# If the ARCH_PATTERN environment variable is not set, the system generates a fatal log and stops the
# compilation.
if(NOT DEFINED ENV{ARCH_PATTERN})
    message(FATAL_ERROR "please define environment variable:ARCH_PATTERN")
endif()
# Find the Base directory in the relative directory of CMakeList.txt and assign the permission to
ASCEND_BASE_DIR.
Set(ASCEND_BASE_DIR ${CMAKE_CURRENT_SOURCE_DIR}/../ascendbase/src/Base)
# Find the absolute path of ASCEND_BASE_DIR and assign it to ASCEND_BASE_ABS_DIR.
get_filename_component(ASCEND_BASE_ABS_DIR ${ASCEND_BASE_DIR} ABSOLUTE)
# Obtain all source files in the ASCEND_BASE_ABS_DIR directory.
file(GLOB_RECURSE ASCEND_BASE_SRC_FILES ${ASCEND_BASE_ABS_DIR}/*.cpp)
# Add compilation options.
add_compile_options(-std=c++11 -fPIE -fstack-protector-all -Werror)
# Do not use rpath during compilation.
set(CMAKE_SKIP_BUILD_RPATH True)
# Set the directory of the CMakeList.txt file to the root directory of the project.
set(PROJECT_SRC_ROOT ${CMAKE_CURRENT_LIST_DIR}/)
# Set the dist directory in the root directory of the project to the directory generated by the executable file.
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${PROJECT_SRC_ROOT}/dist)
# Set the directory of the header files and library files to be contained.
set(ACL_INC_DIR $ENV{ASCEND_HOME}/$ENV{ASCEND_VERSION}/$ENV{ARCH_PATTERN}/include)
set(CSEC_INC_DIR ${DRIVER_HOME}/driver/kernel/libc_sec/include)
set(DRIVER_INC_DIR ${DRIVER_HOME}/driver/kernel ${DRIVER_HOME}/driver/kernel/inc/driver)
set(ACL_LIB_DIR $ENV{ASCEND_HOME}/$ENV{ASCEND_VERSION}/$ENV{ARCH_PATTERN}/lib64)
set(DRV_LIB_DIR ${DRIVER_HOME}/driver/lib64)
# Include the header file during compilation.
include_directories(${DRIVER_INC_DIR})
include_directories(${ACL_INC_DIR})
include_directories(${CSEC_INC_DIR})
include_directories(${PROJECT_SRC_ROOT})
include_directories(${ASCEND_BASE_DIR})
# Query all source files in the AclProcess directory.
aux_source_directory(${PROJECT_SRC_ROOT}/AclProcess SRC_FILES)
# Link the library file during compilation.
link_directories(
    ${DRV_LIB_DIR}
    ${ACL_LIB_DIR})
# Generate the executable file main.
add_executable(main ${SRC_FILES} ${ASCEND_BASE_SRC_FILES} ${PROJECT_SRC_ROOT}/main.cpp)
# The library files linked to the executable file main are ascendcl ascend_hal acl_dvpp c_sec pthread -Wl,-z,relro,-z,now,-z,noxexecstack -pie -s.
target_link_libraries(main ascendcl ascend_hal acl_dvpp c_sec pthread -Wl,-z,relro,-z,now,-z,noxexecstack -pie -s)
9.1 What Do I Do If Model Conversion Takes Too Long When the OS and Architecture Configuration of the Development Environment Is Arm (AArch64)?

For an Arm (AArch64) development environment, to accelerate model conversion, you can use the numactl tool to specify CPU cores for model conversion as follows:

1. Log in to the development environment as the ATC installation user and run the `su root` command to switch to the root user.
2. Ensure that the development environment is connected to the network, and run the following command to install the numactl tool:
   ```bash
   yum -y install numactl
   ``
3. Switch to the ATC installation user and run the `numactl -C` command to specify CPU cores 16–31 to process model conversion:
   ```bash
   numactl -C 16-31 --localalloc <args>
   ``
   Replace `<args>` with the actual ATC model conversion command.

CPU cores 16–31 are recommended for better processing performance. You can also change the CPU cores as required.
## Change History

<table>
<thead>
<tr>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2020-07-22</td>
<td>This issue is the first official release.</td>
</tr>
</tbody>
</table>