

A child wearing a pilot's cap and goggles sits on the shoulder of a large, white, humanoid robot. The child is pointing their finger towards a large, glowing globe in the background. The globe features a stylized world map overlay. The scene is set against a bright blue sky with streaks of light, suggesting a futuristic or space-themed environment.

AscendCL模型推理动态特性

课程目标

- 学完本课程，您应该能够：
 - 掌握AscendCL模型推理动态系列特性，包括动态AIPP、动态Batch、动态分辨率以及动态维度等功能
- 为了达成上述目标，您应该具备如下知识：
 - 熟练的C/C++语言编程能力
 - ATC模型转换工具用法
 - AscendCL基础功能

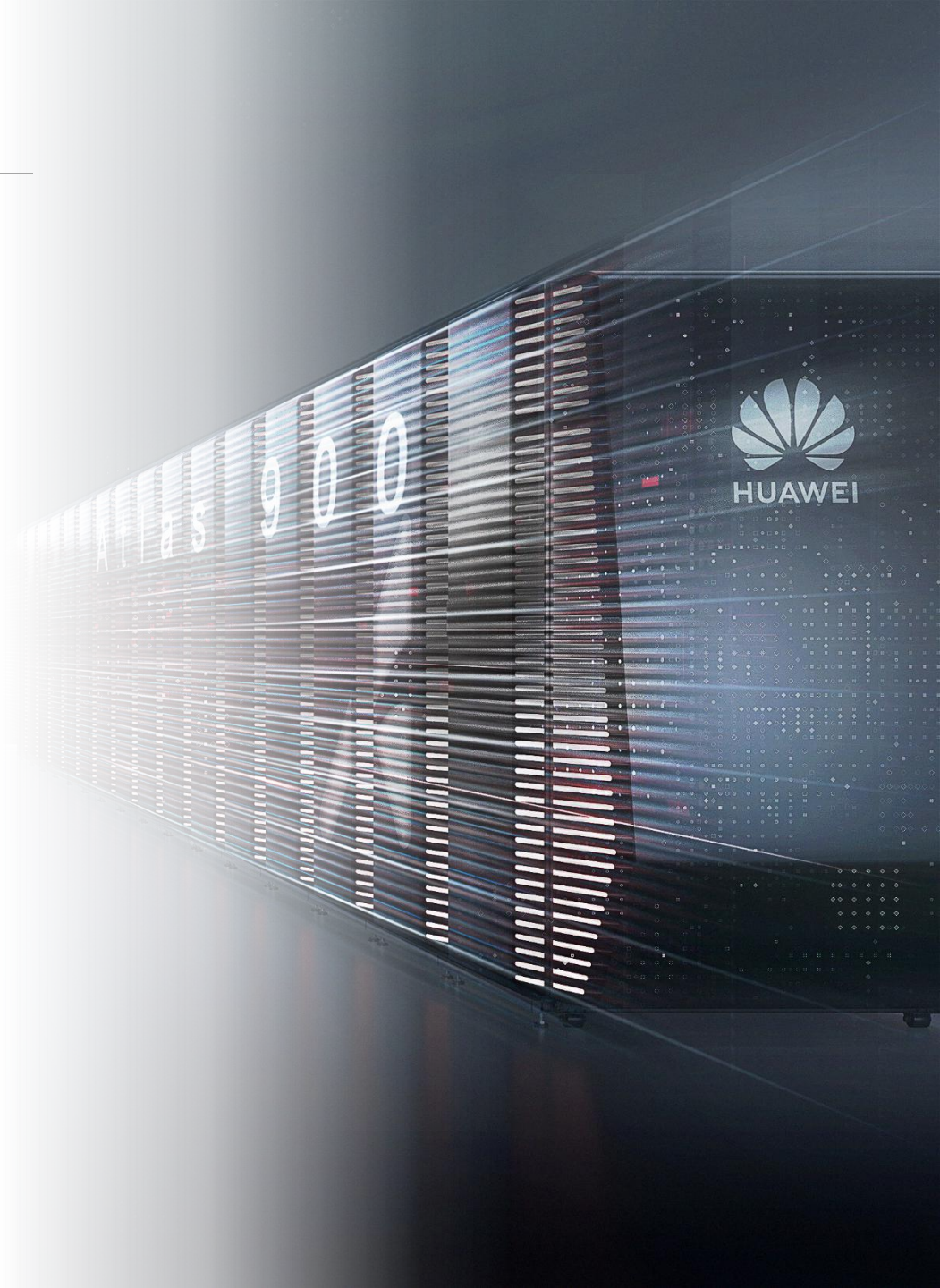
1 动态特性概述

2 动态AIPP

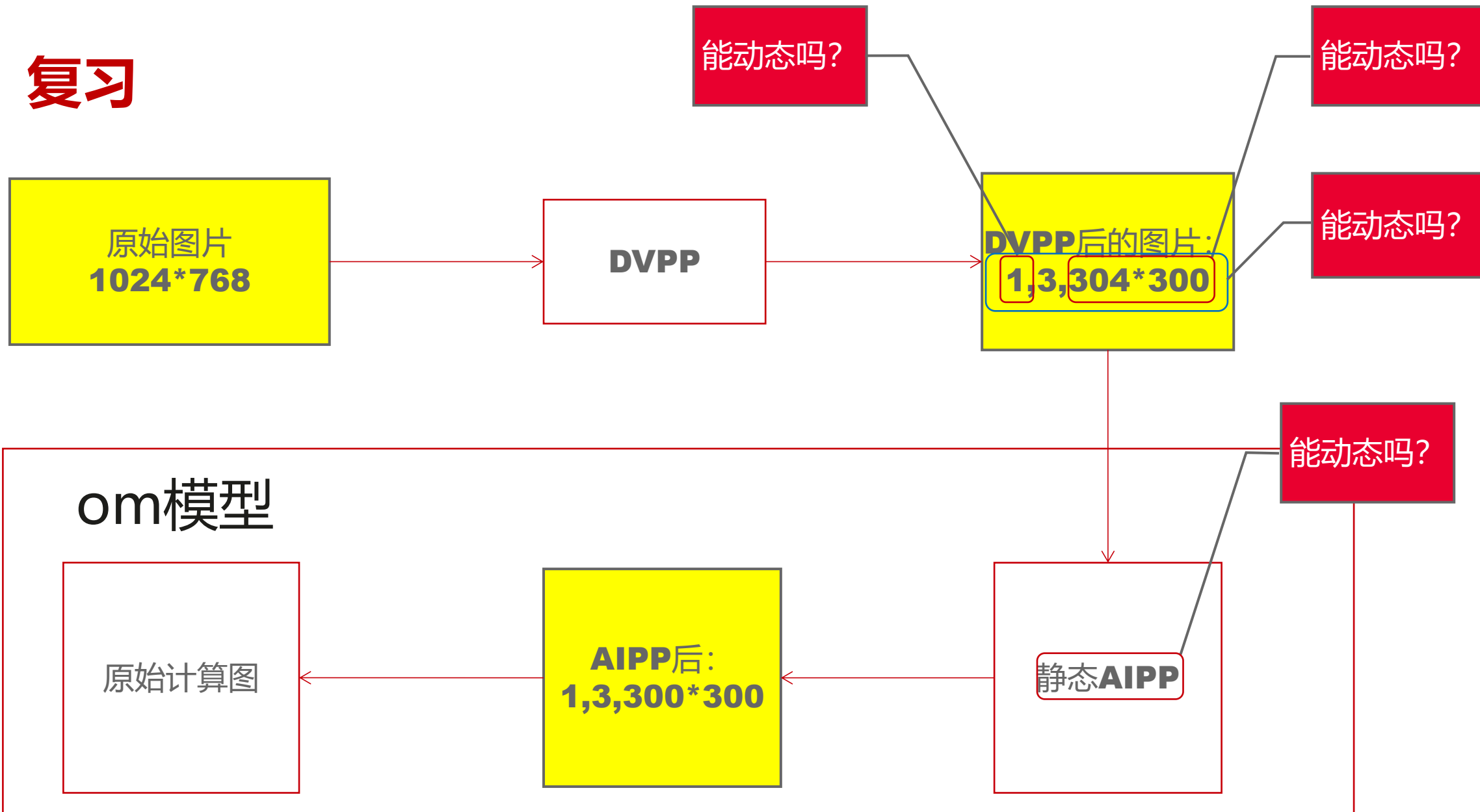
3 动态Batch

4 动态分辨率

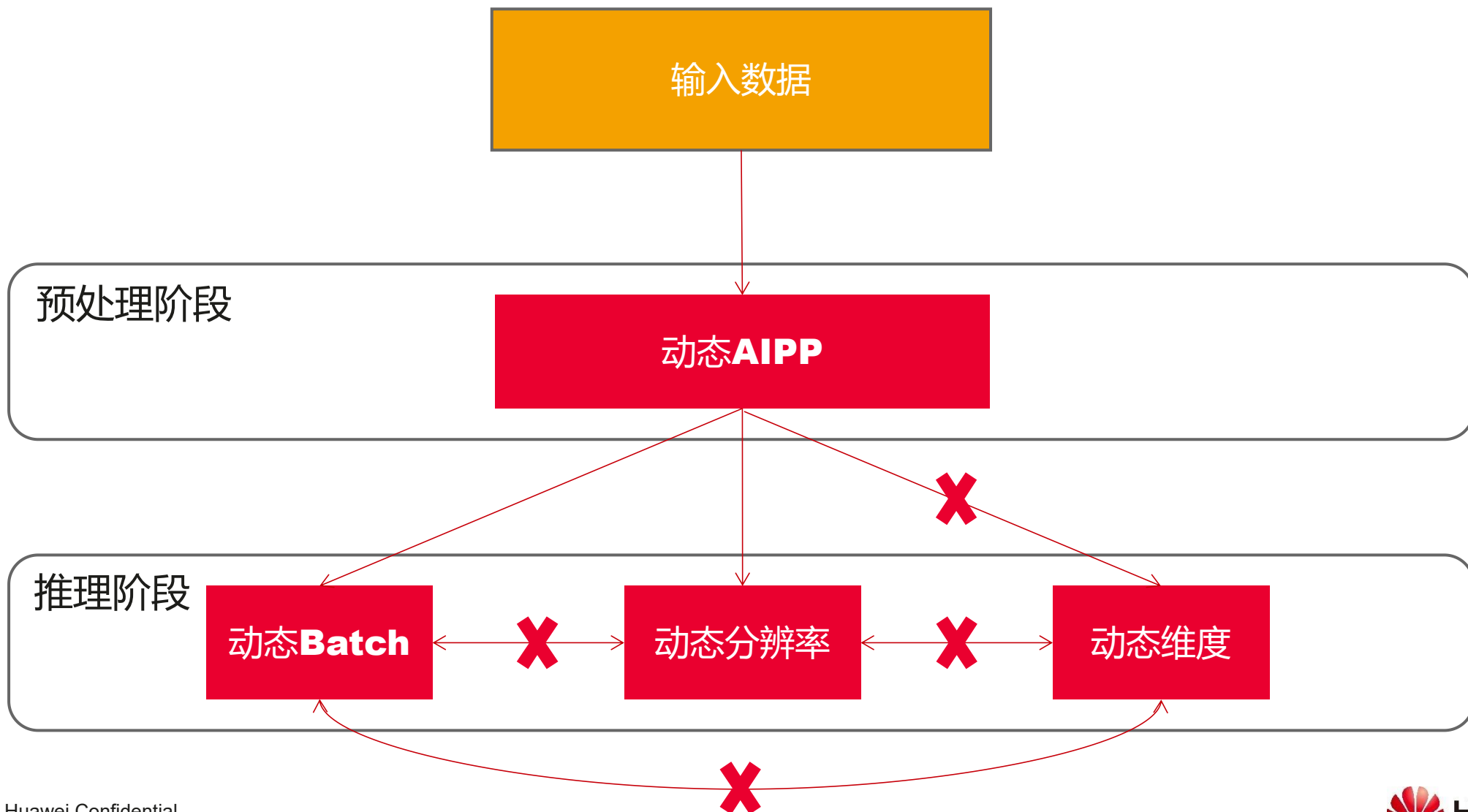
5 动态维度



复习



四个动态的兼容性



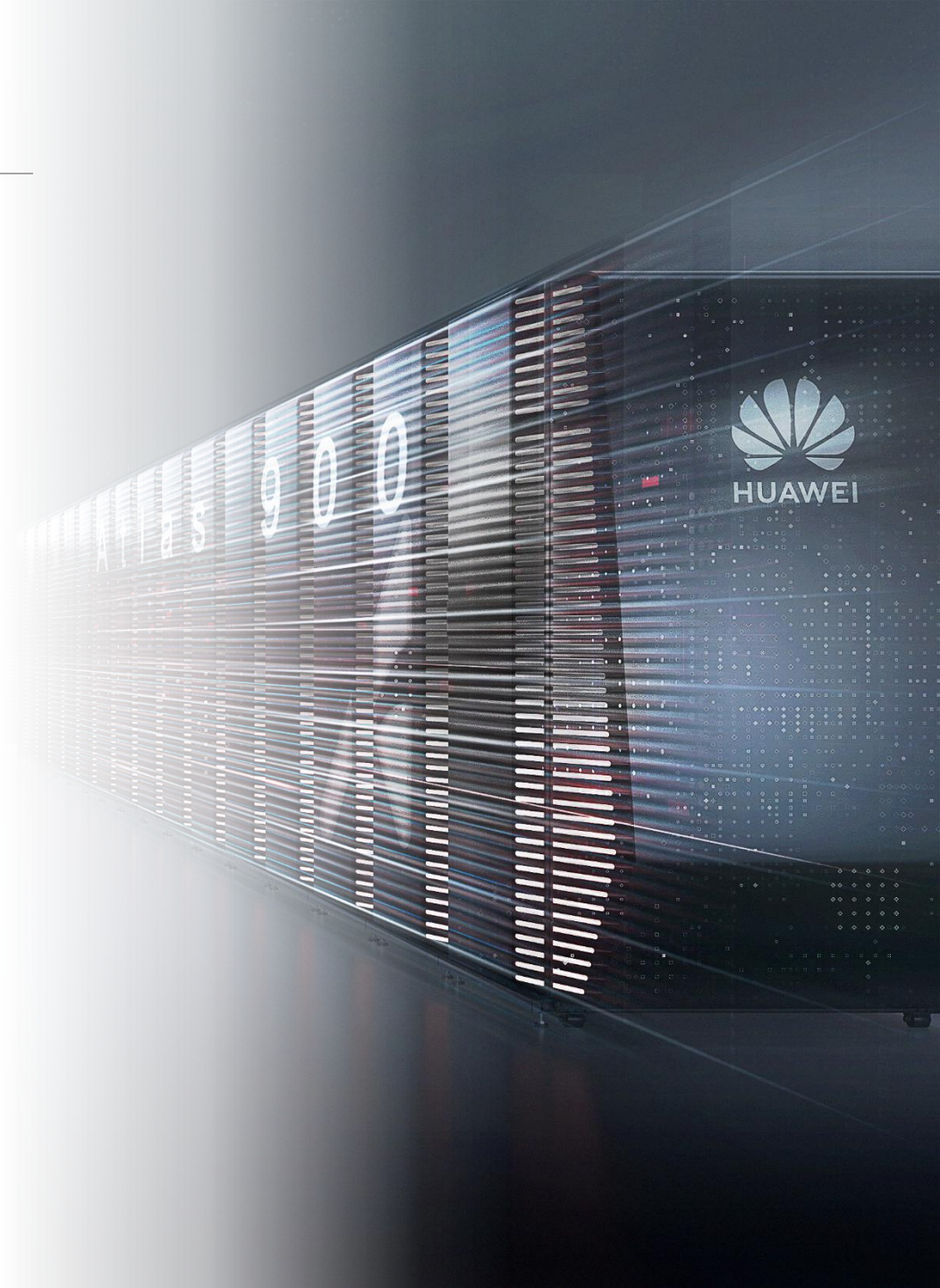
1 动态特性概述

2 动态AIPP

3 动态Batch

4 动态分辨率

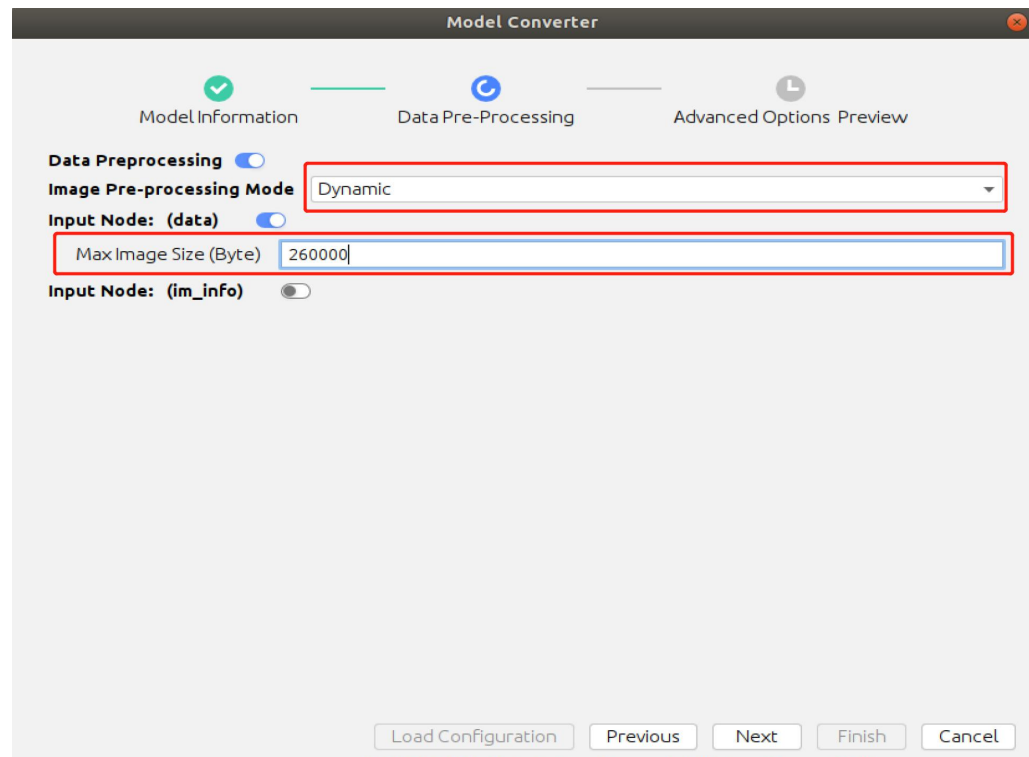
5 动态维度



动态AIPP –准备阶段

选用静态**AIPP**时，**AIPP**功能是通过模型转换工具做进**om**模型中的。而在动态**AIPP**场景下，这些预处理操作都可以通过编程来完成。这也就给了我们在运行时动态调整**AIPP**参数的能力。

但是在选用动态**AIPP**功能时，要在模型转换过程中做一些特殊处理：



动态AIPP – Coding

由于模型的输入输出大小通常比较固定，所以实践中经常较早申请模型的输入输出内存，也就是准备Dataset和DataBuffer，在这个阶段，动态AIPP是有一些特殊处理的；而通常在实际执行推理前才能够确定要做怎样的预处理，所以在推理调用前才设置实际的AIPP参数。

故，我们把动态AIPP特性分成两阶段的调用：

- 内存准备阶段
- 模型推理阶段



动态AIPP – Coding

内存准备阶段:

1. 调用**`aclError aclmdlGetInputIndexByName(const aclmdlDesc *modelDesc, ACL_DYNAMIC_AIPP_NAME, size_t *index)`**

以获取动态AIPP配置输入是在模型输入的第几个位置（不一定在最后一个）

2. 调用**`size_t aclmdlGetInputSizeByIndex(aclmdlDesc *modelDesc, size_t index)`**以获取这个位置的输入需要多大一块内存

3. 调用**`aclrtMalloc`**申请上一步结果大小的device内存（不要往里边写任何东西），并用这块device内存构造一个**`DataBuffer`**

4. 在构造**`Dataset`**的时候把这个**`DataBuffer`**放置到**`index`**指引的位置上

（**`aclmdlAddDatasetBuffer`**这个接口是按顺序添加**`DataBuffer`**的，没有**`index`**参数，所以怎么把**`Buffer`**放到指定位置上，发挥一下想象力:）

动态AIPP – Coding

模型推理阶段：

1. 调用**`aclError aclmdlGetInputIndexByName(const aclmdlDesc *modelDesc, ACL_DYNAMIC_AIPP_NAME, size_t *index)`**

以获取动态AIPP配置输入是在模型输入的第几个位置

2. 创建一个动态AIPP配置对象，并设置参数值：

```
aclmdlAIPP *aippDynamicSet = aclmdlCreateAIPP(batchNumber);  
    ret = aclmdlSetAIPPSrcImageSize(aippDynamicSet, 256, 224);  
    ret = aclmdlSetAIPPInputFormat(aippDynamicSet, ACL_YUV420SP_U8);  
    ret = aclmdlSetAIPPCscParams(aippDynamicSet, 1, 256, 443, 0, 256, -86, -  
178, 256, 0, 350, 0, 0, 0, 0, 128, 128);  
    ret = aclmdlSetAIPPRbuvSwapSwitch(aippDynamicSet, 0);  
    ret = aclmdlSetAIPPDtcPixelMean(aippDynamicSet, 0, 0, 0, 0, 0);  
    ret = aclmdlSetAIPPDtcPixelMin(aippDynamicSet, 0, 0, 0, 0, 0);  
    ret = aclmdlSetAIPPPixelVarReci(aippDynamicSet, 1, 1, 1, 1, 0);  
    ret = aclmdlSetAIPPCropParams(aippDynamicSet, 1, 0, 0, 224, 224, 0);
```

3. 把这个动态AIPP配置对象放到模型中（放完就可以销毁了）：

```
ret = aclmdlSetInputAIPP(modelId_, input_, index, aippDynamicSet);  
ret = aclmdlDestroyAIPP(aippDynamicSet);
```

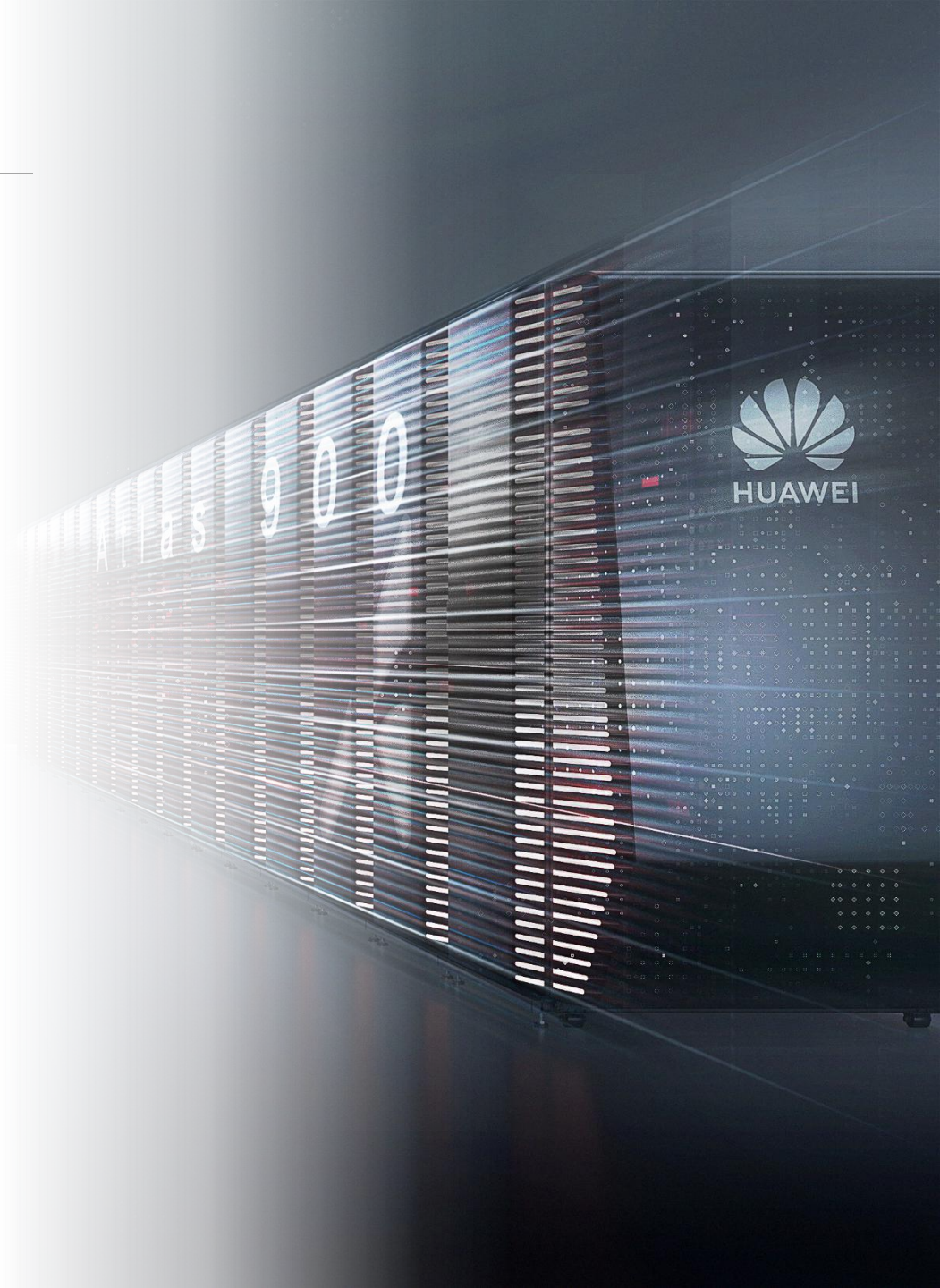
1 动态特性概述

2 动态AIPP

3 动态Batch

4 动态分辨率

5 动态维度

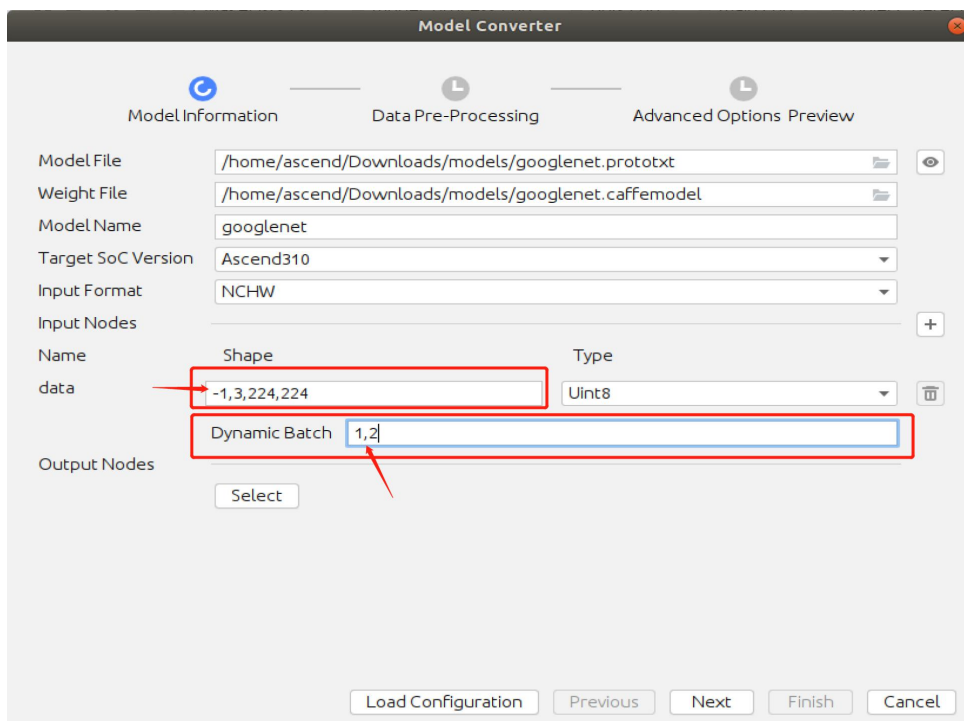


动态Batch – 准备阶段

如果在执行推理业务时，每次处理的图片数量不固定，则可以通过配置动态**Batch**参数来动态分配每次处理的图片数量。

例如用户执行推理业务时需要每次处理**2张，4张，8张**图片，则需要在转换模型时配置上指定**2,4,8batch**档位，申请了档位后，模型推理时需要根据实际档位申请内存。

注意，这里所说的“动态”**Batch**，是有限的动态，要设置有限个数的档位，最大**1000**档，而非任意**Batch**数都能处理。

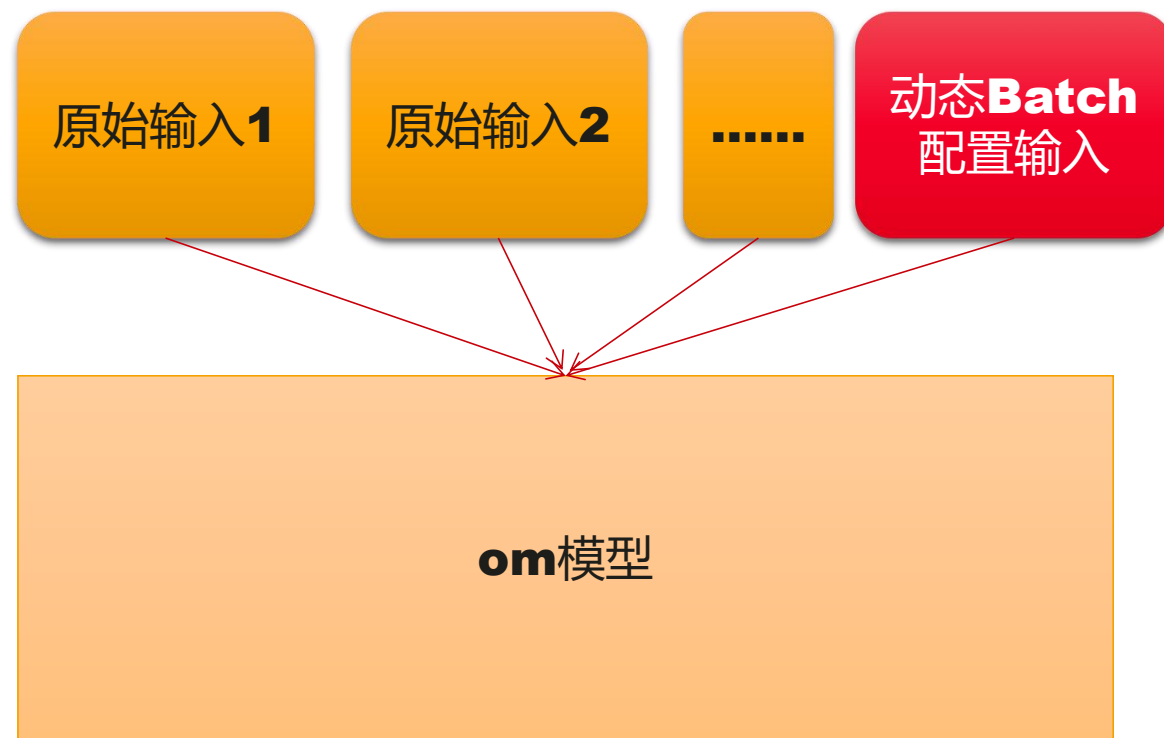


动态Batch – Coding

在“多一个输入”方面，几个动态特性是一样的，动态Batch也是多一个输入。

故，我们把动态Batch特性分成两阶段的调用：

- 内存准备阶段
- 模型推理阶段



动态Batch- Coding

内存准备阶段:

1. 调用**`aclError aclmdlGetInputIndexByName(const aclmdlDesc *modelDesc, ACL_DYNAMIC_TENSOR_NAME, size_t *index)`**

以获取动态AIPP配置输入是在模型输入的第几个位置（不一定在最后一个）

2. 调用**`size_t aclmdlGetInputSizeByIndex(aclmdlDesc *modelDesc, size_t index)`**以获取这个位置的输入需要多大一块内存

3. 调用**`aclrtMalloc`**申请上一步结果大小的device内存（不要往里边写任何东西），并用这块device内存构造一个**`DataBuffer`**

4. 在构造**`Dataset`**的时候把这个**`DataBuffer`**放置到**`index`**指引的位置上

（**`aclmdlAddDatasetBuffer`**这个接口是按顺序添加**`DataBuffer`**的，没有**`index`**参数，所以怎么把**`Buffer`**放到指定位置上，发挥一下想象力:）

动态Batch- Coding

模型推理阶段：

1. 调用 **`aclError aclmdlGetInputIndexByName(const aclmdlDesc *modelDesc, ACL_DYNAMIC_TENSOR_NAME, size_t *index)`**

以获取动态**Batch**配置输入是在模型输入的第几个位置

2. 告诉模型，本次推理的**Batch**数是多少：

```
uint64_t batchSize = 8;
```

```
ret = aclmdlSetDynamicBatchSize(modelId_, input_, index, batchSize);
```

3. 正常推理就可以了

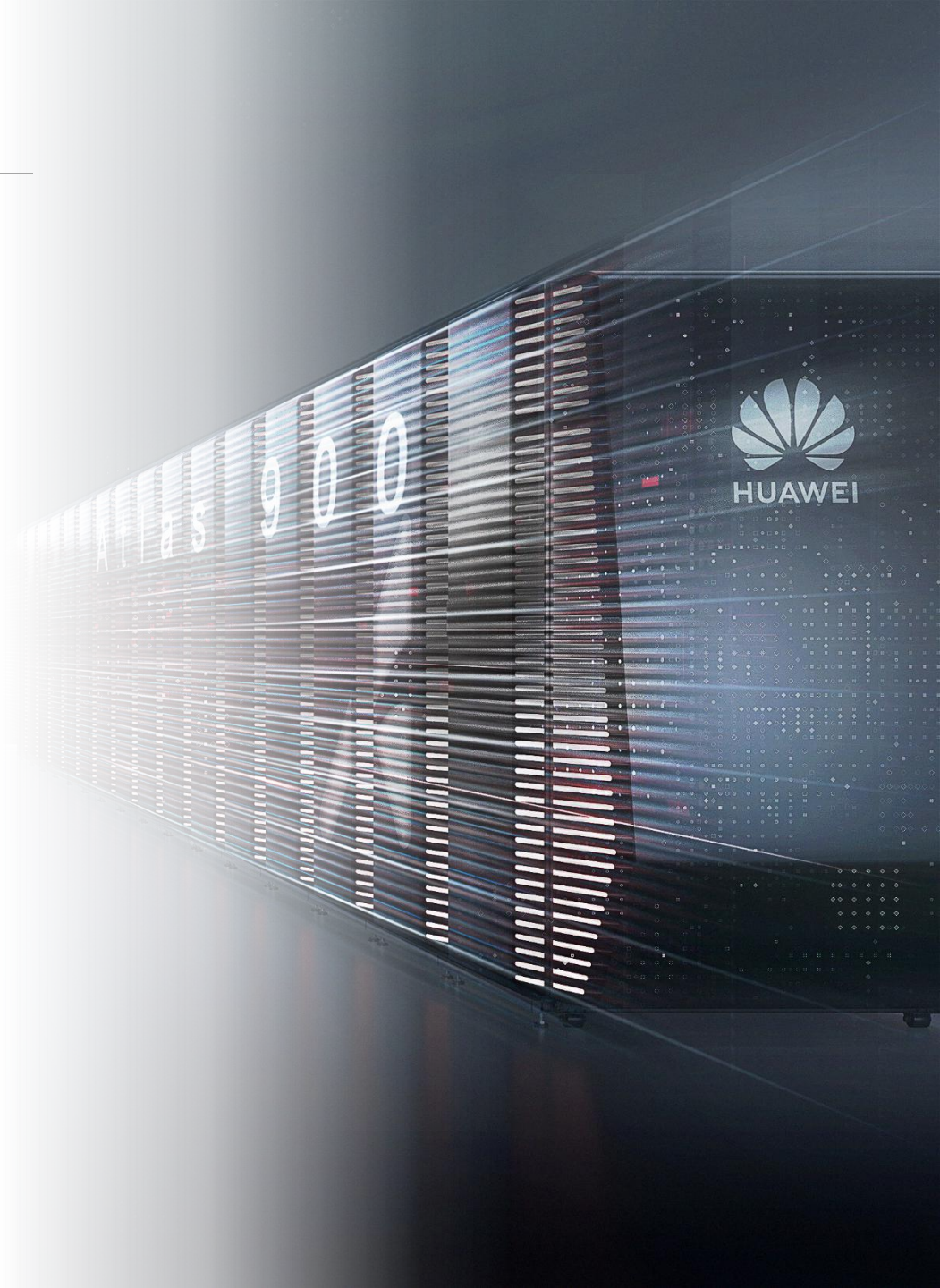
1 动态特性概述

2 动态AIPP

3 动态Batch

4 动态分辨率

5 动态维度



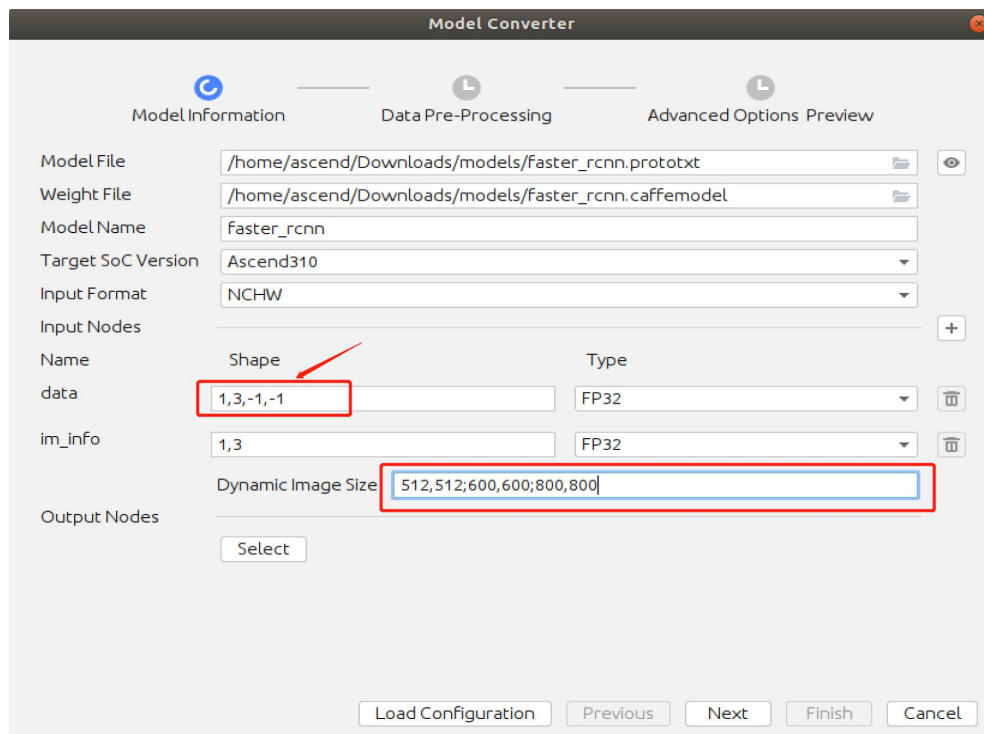
动态分辨率 – 准备阶段

如果在执行推理业务时，每次处理的图片分辨率不固定（比如超分场景），则可以通过配置动态分辨率参数来动态分配每次处理的图片分辨率。

注意动态分辨率功能不能和动态**Batch**、动态维度功能混用。

例如用户执行推理业务时需要（**224,224**）或（**448,448**）分辨率的图片，则需要在转换模型时配置上指定（**224,224;448,448**）档位，申请了档位后，模型推理时需要根据实际档位申请内存。

注意，这里所说的“动态”分辨率，是有限的动态，要设置有限个数的档位，最大**1000**档，而非任意分辨率都能处理。



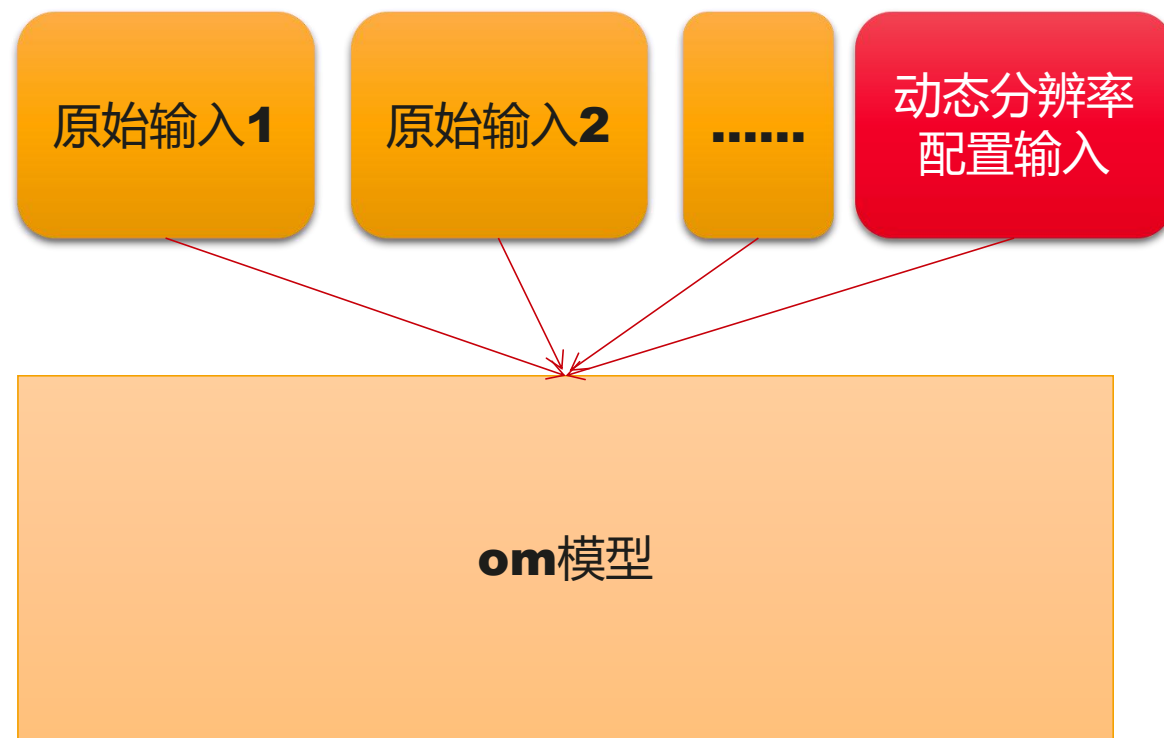
The screenshot shows the 'Model Converter' application window. The 'Data Pre-Processing' tab is selected. The 'Model File' is set to '/home/ascend/Downloads/models/faster_rcnn.prototxt' and the 'Weight File' is '/home/ascend/Downloads/models/faster_rcnn.caffemodel'. The 'Model Name' is 'faster_rcnn'. The 'Target SoC Version' is 'Ascend310' and the 'Input Format' is 'NCHW'. Under 'Input Nodes', the 'data' node has a 'Shape' of '1,3,-1,-1' (highlighted with a red box) and a 'Type' of 'FP32'. The 'im_info' node has a 'Shape' of '1,3' and a 'Type' of 'FP32'. The 'Dynamic Image Size' field is set to '512,512;600,600;800,800' (highlighted with a red box). The 'Output Nodes' section has a 'Select' button. At the bottom, there are buttons for 'Load Configuration', 'Previous', 'Next', 'Finish', and 'Cancel'.

动态分辨率 – Coding

在“多一个输入”方面，几个动态特性是一样的，动态分辨率也是多一个输入。

故，我们把动态分辨率特性分成两阶段的调用：

- 内存准备阶段
- 模型推理阶段



动态分辨率 – Coding

内存准备阶段:

1. 调用**`aclError aclmdlGetInputIndexByName(const aclmdlDesc *modelDesc, ACL_DYNAMIC_TENSOR_NAME, size_t *index)`**

以获取动态AIPP配置输入是在模型输入的第几个位置（不一定在最后一个）

2. 调用**`size_t aclmdlGetInputSizeByIndex(aclmdlDesc *modelDesc, size_t index)`**以获取这个位置的输入需要多大一块内存

3. 调用**`aclrtMalloc`**申请上一步结果大小的device内存（不要往里边写任何东西），并用这块device内存构造一个**`DataBuffer`**

4. 在构造**`Dataset`**的时候把这个**`DataBuffer`**放置到**`index`**指引的位置上

（**`aclmdlAddDatasetBuffer`**这个接口是按顺序添加**`DataBuffer`**的，没有**`index`**参数，所以怎么把**`Buffer`**放到指定位置上，发挥一下想象力:）

动态分辨率 – Coding

模型推理阶段：

1. 调用 **`aclError aclmdlGetInputIndexByName(const aclmdlDesc *modelDesc, ACL_DYNAMIC_TENSOR_NAME, size_t *index)`**

以获取动态分辨率配置输入是在模型输入的第几个位置

2. 告诉模型，本次推理的分辨率是多少：

```
uint64_t height = 224;
```

```
uint64_t width = 224;
```

```
ret = aclmdlSetDynamicHWSIZE(modelld_, input_, index, height, width);
```

3. 正常推理就可以了

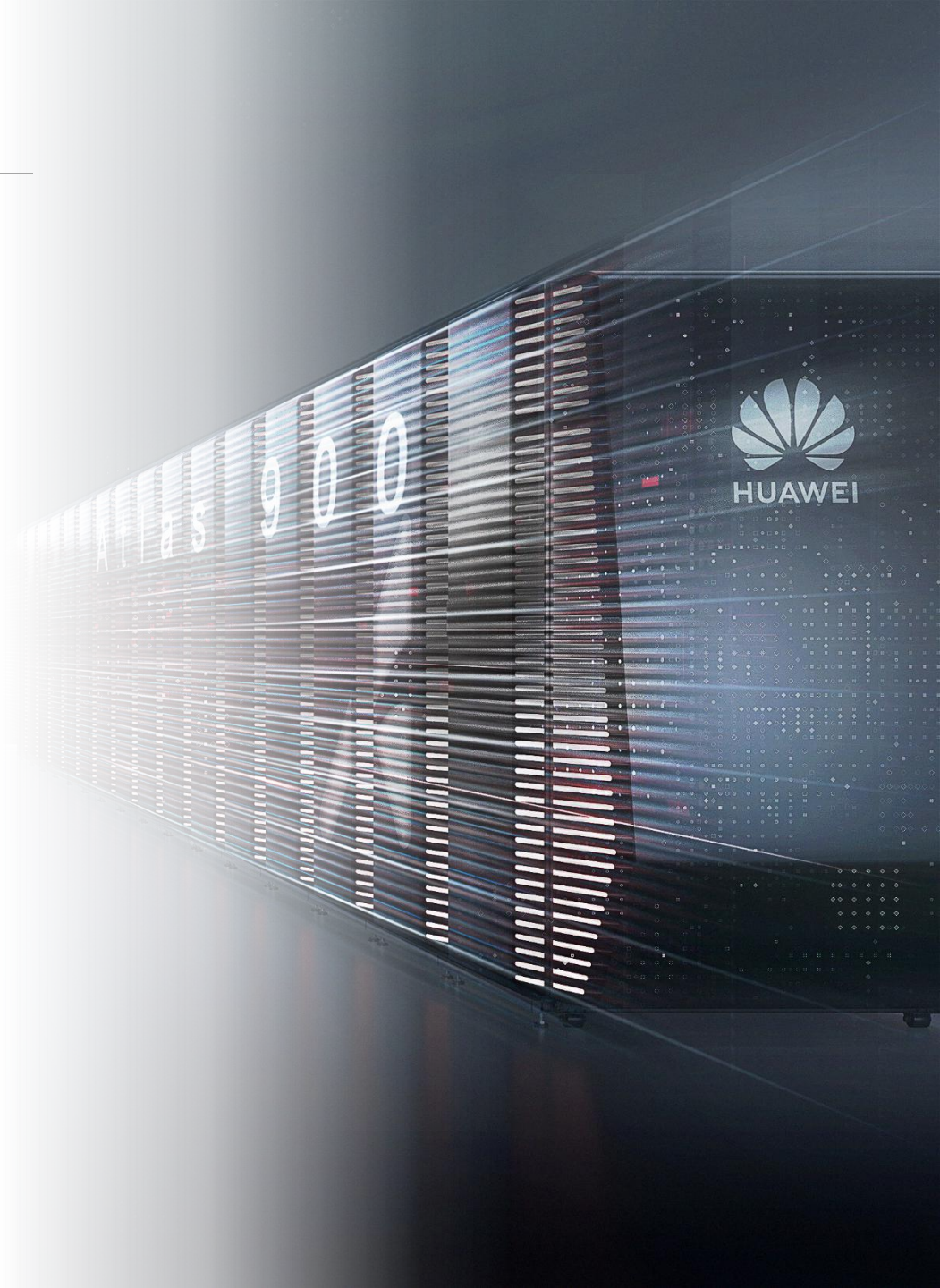
1 动态特性概述

2 动态AIPP

3 动态Batch

4 动态分辨率

5 动态维度



动态维度 – 准备阶段

某些推理场景并非计算视觉类，比如**Transformer**，这种场景下，数据的维度数量就不一定是**4**，或者说排布格式就不一定是**NCHW**了。此类数据的维度可能较高，而如果在这种场景下仍需要动态决定输入数据尺寸，则需要用到**AscendCL**的动态维度特性。

注意动态维度功能不能和其他任何动态特性（动态**Batch**、动态分辨率、动态**AIPP**）功能混用。

根据实际业务需要，在模型转换阶段需要先配置动态维度的信息：

```
ascend@ubuntu:~/Downloads/models$ atc --model=faster_rcnn.prototxt --weight=faster_rcnn.caffemodel --framework=0 --output=faster_rcnn
--soc_version=Ascend310 --input_shape="data:1,3,-1,-1;im_info:1,3" --dynamic_dims="512,512;600,600;800,800" --input_format=ND
ATC start working now, please wait for a moment.
ATC run success, welcome to the next use.
```

动态维度 – Coding

在“多一个输入”方面，几个动态特性是一样的，动态维度也是多一个输入。

故，我们把动态维度特性分成两阶段的调用：

- 内存准备阶段
- 模型推理阶段



动态维度 – Coding

内存准备阶段:

1. 调用**`aclError aclmdlGetInputIndexByName(const aclmdlDesc *modelDesc, ACL_DYNAMIC_TENSOR_NAME, size_t *index)`**

以获取动态AIPP配置输入是在模型输入的第几个位置（不一定在最后一个）

2. 调用**`size_t aclmdlGetInputSizeByIndex(aclmdlDesc *modelDesc, size_t index)`**以获取这个位置的输入需要多大一块内存

3. 调用**`aclrtMalloc`**申请上一步结果大小的device内存（不要往里边写任何东西），并用这块device内存构造一个**`DataBuffer`**

4. 在构造**`Dataset`**的时候把这个**`DataBuffer`**放置到**`index`**指引的位置上

（**`aclmdlAddDatasetBuffer`**这个接口是按顺序添加**`DataBuffer`**的，没有**`index`**参数，所以怎么把**`Buffer`**放到指定位置上，发挥一下想象力:）

动态维度 – Coding

模型推理阶段：

1. 调用**`aclError aclmdlGetInputIndexByName(const aclmdlDesc *modelDesc, ACL_DYNAMIC_TENSOR_NAME, size_t *index)`**

以获取动态维度配置输入是在模型输入的第几个位置

2. 告诉模型，本次推理的维度信息：

```
aclmdlIODims currentDims;
```

```
currentDims.dimCount = 4;
```

```
currentDims.dims[0] = 8;
```

```
currentDims.dims[1] = 3;
```

```
currentDims.dims[2] = 224;
```

```
currentDims.dims[3] = 224;
```

```
ret = aclmdlSetInputDynamicDims(modelId_, input_, index, &currentDims);
```

3. 正常推理就可以了

注意：1. 这里要放置模型所有输入的所有维度信息，而非像动态Batch和动态分辨率那样只放动态的信息；
2. 所以输入的所有维度数量加到一起不要超过128；所有动态维度数量加到一起不要超过4

Thank you.

昇腾开发者社区



<http://ascend.huawei.com>

把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。

**Bring digital to every person, home, and
organization for a fully connected,
intelligent world.**

**Copyright©2020 Huawei Technologies Co., Ltd.
All Rights Reserved.**

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.

